

A Parallel Interval Computation Model with Alternative Message Passing

Yong Wu*, Arun Kumar†, and Peng Shi*

*Institute for Logistics and Supply Chain Management, Victoria University, Melbourne Australia

Email: wuyong@pmail.ntu.edu.sg, {yong.wu, peng.shi}@vu.edu.au

†School of Aerospace, Mechanical & Manufacturing Engineering, RMIT University, Melbourne Australia

Email: a.kumar@rmit.edu.au

Abstract—In this paper, we propose a decentralized parallel computation model for global optimization using interval analysis. The model is adaptive to any number of processors and there is no need to design an initial decomposition scheme to feed each processor at the beginning. The work load is distributed evenly among all processors by alternative message passing. Numerical experiments indicate that the model works well and is stable with different number of parallel processors, distributes the load evenly among the processors, and provides an impressive speedup, especially when the problem is time-consuming to solve.

Keywords—global optimization; interval analysis; parallel processing; computation model; branch-and-bound;

I. INTRODUCTION

The global optimization problem over the search domain $\mathcal{D} \subset \mathbb{R}^n$ which is defined by the lower and upper bounds of all variables, is considered in this research:

$$\begin{aligned} \max \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, r \\ & h_j(\mathbf{x}) = 0, \quad j = r + 1, \dots, s \\ & \mathbf{x} \in \mathcal{D}. \end{aligned} \quad (1)$$

where $f : \mathcal{D} \rightarrow \mathbb{R}$ is the objective function, $g_i : \mathcal{D} \rightarrow \mathbb{R}$, $i = 1, \dots, r$ and $h_j : \mathcal{D} \rightarrow \mathbb{R}$, $j = r + 1, \dots, s$ are the inequality and equality constraints, respectively.

Due to the nonlinearity of the objective function and constraints expressed in (1), some effort must be spent if global optimal solutions are wanted. Interval methods [1], [2] are partitioning optimization techniques which have the advantage of providing explicit ranges over functions to guarantee assured interval disposal. A box (hypercube) can be discarded or removed from consideration when it is verified that there are no better solutions inside it or it is infeasible, or the objective function range over the box is less than some small value, ε_f [2].

Branch-and-bound is normally applied to interval analysis. Boxes are selected according to some strategies: some have long been used in interval methods, such as the strategy to select the box with the best upper bound [3], select the box which is the oldest [4], and select the box which has maximum width [4] and so on. Once the box is selected, it will be partitioned along one or several of its edges [1], such as subdividing along the widest edge [5]. There are other

rules that utilize the gradient, such as the rule of Hansen and Walster [1] and the rule of Ratz [6]. More recently, some heuristic selection strategies have been proposed in [7]–[9].

However, it should be noted that the number of boxes to be managed usually increases exponentially. This requires both more efficient box disposal procedures and more powerful box management schemes and computation models. Parallel processing can be applied to speedup the computation. However, due to the fact that no knowledge of the execution behavior is known *a priori*, it is not suitable to statically balance the load at the very beginning. Further, it is difficult to achieve an optimal decomposition for branch-and-bound problems even when static load balance schemes are feasible [10]. Therefore, parallel computation models must address the load balancing problem.

Master/Slave architecture (centralized model), such as in [11], has the advantage of efficiently utilizing available information. However, the master processor might become a bottleneck for the whole computation model due to data management and communication, especially when the number of processors is large. Therefore, it is reasonable for each processor to store a fraction of the boxes to be processed. Different load balancing mechanisms have been proposed. Such as, Eriksson and Lindstrom [12] investigated dynamic load balance schedulers. Benyoub and Daoudi [13] proposed an approach based on synchronization and uniform redistribution of boxes. Ibraev [14] presented a challenge leadership model where each processor has the chance to become the “leader” when it finds better solution. Gau and Stadtherr [15] provided a stage-wise asynchronous work load information exchange strategy for dynamic load balancing. Berner [16] presented a parallel method using a centralized mediator for the dynamic load balancing. Usually, load balancing make the management of communication a complex task for code implementation.

In this study, we propose a decentralized parallel computation model for interval analysis. The model does not need an initial decomposition scheme to feed every processor a subproblem at the beginning, and there is no need to consider the problem of load balancing. These characteristics make the parallel implementation almost as simple as a serial one. The load is automatically balanced by alternative message passing operations among all processors. We refer

this approach as PBM (Parallel Branch-and-bound Model).

The remainder of this paper is organized as follows: in Section II, the PBM model is introduced and the model properties are analyzed; Section III conducts numerical experiments to show the performance of PBM and Section IV concludes this paper.

II. THE PARALLEL MODEL

A. Model Topology

All processors in PBM are arranged in a ring topology. Suppose there are totally p processors, each processor is assigned a *rank* ranging from 0 to $p-1$. Messages are passed counterclockwise along the ring, but are not necessarily sent and received between directly connected neighbors, e.g. from P_1 to P_2 . Instead, messages are sent to and received from processors which are a hop away from the sending processors. The hop, h , changes in a periodic way according to a series of $2^0, 2^1, 2^2, \dots, 2^{C-1}$, where $C = \text{round}(\log_2 p)$. Thus, one particular processor sends messages to a group of processors alternatively, this is referred as the alternative message passing of PBM.

The rank of the processor to which processor P_i should send message can be represented as $(i+h) \bmod p$, and the rank of the processor from which a message can be received can be denoted as $(i-h+p) \bmod p$. Taking a parallel environment with 8 processors as an example, the alternative message passing can be represented in Fig. 1. The operations with $h = 1, 2, 4$ constitute a complete cycle of alternative message passing since $C = 3$ when there are 8 processors.

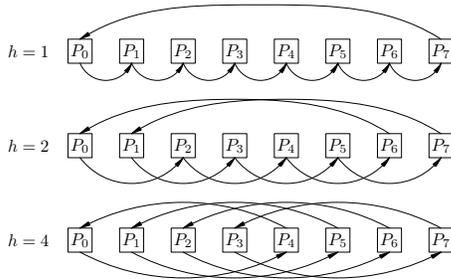


Figure 1. The alternative message passing.

B. The Model

In PBM, each processor maintains a binary tree T to manage the subproblems (branches). Initially, PBM sets the whole search space as a single problem and stores it at processor P_0 , the tree T on other processors are empty at the beginning. We define *computation cycle* as the cycle from one message passing operation to the next same message passing operation for ease of description.

At the beginning of each computation cycle, the status of the tree T is checked. If T is not empty, a subproblem is retrieved from it for sending; otherwise, an empty message is prepared. Then messages are sent by non-blocking

sending operations to destination processors and followed by blocking receiving operations from source processors and subsequently messages are processed. Here, by performing non-blocking sending and blocking receiving, the communication time can be minimized. It should also be pointed out that sending an empty message is just to keep the message flow continuous and the order of alternative message passing consistent with other processors.

Consequently, a subproblem is selected from T if it is not empty. The problem is treated in a basic branch-and-bound way and subdivided into subproblems according to the branching rules. Each subproblem is then evaluated, and either inserted back into T or discarded if it meets the disposal criteria.

When a better solution is found in current computation cycle or a processor's tree T is empty, an 'all gathering' operation is triggered. The operation collects known best solutions from each processor to update the tree T and the status of each processor to decide whether the procedure satisfies the termination criterion. The whole computation is terminated when all processors are idle.

C. Model Properties

We use the concept *dominant* to describe a node (subproblem) that is to be selected for processing. A *local dominant* node is the node which can be selected according to the selection rules among all nodes on the local processor. A *global dominant* node is the node which can be selected according to the selection rules among all nodes on *all* processors.

In PBM, each processor first sends its local dominant node to another processor and processes the local dominant node after the message received has been handled. This makes the dissemination of globally dominant nodes rather easily achievable since a local dominant node on one processor is not necessarily local dominant on another. Such exchanges of local dominant nodes ensure the most prominent nodes are processed first and the order of nodes to be treated is similar to a serial implementation, and therefore, avoids unnecessary subproblem evaluations.

The properties of PBM utilizing p processors can be summarized as follows:

- If a node is globally dominant and its child nodes are locally dominant on each corresponding processor provided no child node can be discarded, then child nodes split from this globally dominant node will dominate all processors after $\text{round}(\log_2 p)$ computation cycles.
- A processor can only be idle for at most $\text{round}(\log_2 p)$ computation cycles consecutively if there is at least one node left among all processors and the child nodes split from that node cannot be discarded.
- If all processors are not idle, computation cycles on each processor will cost almost the same amount of time and the same data managing complexity.

III. NUMERICAL EXPERIMENTS

A. Test problems

Ten maximization test problems are selected to evaluate PBM's performance. We focus on the computational performance of the parallel model and select the box with best upper bound for processing and select the branching direction according to the maximum constraint violation. Detailed description of the procedure can be found in [17].

Table I
SUMMARY OF TEST PROBLEMS.

Problem	Dim.	Constraints	Source
1	5	3 equality	Sect. 8.2.7 in [18]
2	6	4 equality, 1 inequality	Sect. 8.2.8 in [18]
3	5	6 inequality	Sect. 7.2.5 in [18]
4	4	1 equality, 1 inequality	Rastrigin(4) + 2 constraints
5	8	6 inequality	Sect. 5.4.2 in [18]
6	6	4 equality, 1 inequality	Sect. 7.2.2 in [18]
7	6	6 inequality	Sect. 3.4 in [18]
8	8	1 equality, 1 inequality	Rastrigin(8) + 2 constraints
9	10	1 equality, 1 inequality	Rastrigin(10) + 2 constraints
10	20	14 equality	CUTEr [19]

Table I summarizes the dimensions, type and number of constraints, and the reference of the 10 problems. Details of the test problems can be found in [18] and [19], except the Rastrigin problems, which are presented in Appendix A.

B. Test environment

The parallel computation model is tested in a cluster with up to 16 processors (Intel^R XeonTM 3.06 GHz). We tested the performance of PBM with 1, 2, 4, 8, 16 processors. The message passing operations are done by MPI [20].

C. Results

Table II presents the CPU time in seconds. We only present the time used by PBM in this paper since the computational performance is the goal and due to the limited space. For Problems 1, 2 and 3, the time used by the parallel implementation is less than the serial model for at least once. However, the time spent increases when shifting from 8 to 16 processors for Problem 1 and 3; and from 4 processors thereafter for Problem 2. This suggests that the additional cost spent on communication is greater than the gain by taking more processors in these problems. For Problems 4 to 10, the time used decreases monotonically as the number of parallel processors increases.

Of course, in practical computation, there is no need (or not worth) to apply parallel computation to problems that can be solved by a sequential algorithm in a few seconds, such as Problems 2–5. However, the results are included here to demonstrate that the performance of the parallel computation model is not affected too much considering the overhead of parallel initialization and message passing. We can observe that for problems which require longer time to process, super-linear speedup can be achieved.

Table II
CPU TIME USED (IN SECONDS).

Problem	1	2	4	8	16
1	50.4	13.8	6.9	3.8	10.7
2	5.7	1.2	0.7	0.8	1.8
3	2.9	1.4	1.2	1.1	1.9
4	4.3	1.5	0.4	0.2	0.2
5	4.7	1.4	0.5	0.3	0.2
6	496.2	70.9	21.5	13.7	8.1
7	596.0	122.0	37.1	22.9	21.5
8	306.9	91.3	18.6	5.1	4.0
9	11,157.5	3,030.5	480.4	122.0	47.5
10	696.1	248.9	102.8	6.5	0.8
Average	1332.1	358.29	67.0	17.6	9.7

Table III
MAXIMUM BINARY TREE SIZE.

Problem	1	2	4	8	16
1	13,244	6,929	5,911	5,283	5,262
2	3,689	2,417	914	1,432	2,794
3	9,832	6,914	5,235	4,131	6,232
4	7,941	4,044	1,002	346	339
5	2,191	1,942	2,131	2,878	2,033
6	16,146	9,363	4,125	3,912	3,816
7	85,736	41,374	30,603	20,091	17,534
8	87,519	43,380	20,095	4,783	3,499
9	544,816	273,087	91,584	41,134	13,588
10	20,771	15,482	12,485	7,531	802

Table III provides the maximum number of intervals stored (maximum binary tree size) for each problem, which to some extent reflects the workload distribution. It can be observed that the maximum tree sizes are smaller when more processors are utilized for most problems. However, there are some cases where bigger tree sizes are required when using more processors, such as Problems 2, 3 and 5. It is worthwhile to point out that the maximum tree sizes of some problems are more than halved when the number of processors doubles in some occasions. This property is significant when the number of subproblems to be managed is large. Parallel processing reduces the data management complexity, and consequently, reduces the auxiliary time required since the trees are smaller. Moreover, the requirement for memory is also decreased, which is meaningful for processors with limited memory.

D. Discussion

The numerical results presented in this paper indicate that PBM has a high parallel efficiency and is able to provide a decent computational performance. The performance of PBM with more processors is subject to future research and numerical experiments.

The current performance of PBM may be contributed by the following aspects. Firstly, the workload in the parallel implementation could be well-distributed among all processors. The maximum tree size shown in Table III implicitly suggests this. Smaller tree indicates less data management complexity and hence the operations of insertion and dele-

tion of boxes will be quicker. Secondly, each processor sends a local dominant box out and handles its local dominant box after considering the received box. This mechanism helps the model to process high-quality boxes among all processors. Finally, in the serial implementation only the box with the best upper bound can be processed in each iteration; while in parallel implementation, the search space is explored in a different way from that of serial implementation. This alternative order of intervals to be processed might lead to a more efficient way of exploring the search space. At the same time, this effect is more conspicuous when the problem dimension is high since it normally takes longer to split a high dimensional box into disposable small ones.

IV. CONCLUSION

In this research, we proposed a decentralized parallel computation model for branch-and-bound problems and applied it to interval analysis. By performing the alternative message passing with different hop steps, the work load is rapidly and evenly distributed among all processors participating in the computation. The alternative message passing enables the global dominant node and its child nodes spread out all processors within $\text{round}(\log_2 p)$ computation cycles. Numerical experiments indicate that PBM will not only speedup the computation, but also explore the search space in a different and more efficient way, the performance is especially distinct when the problems are difficult and time-consuming to solve.

The model proposed in this paper is applicable to any kind of branch-and-bound problems. Since the problem is treated as a whole single problem at the beginning, no effort is needed to design an initial decomposition for each processor. This simplifies the implementation of PBM since it is difficult to find an optimal initial decomposition scheme in practice. In addition, the communication between processors is easy to control and the parallel model is easy to implement.

APPENDIX A.

TEST PROBLEMS OF RASTRIGIN

Rastrigin(n) [21], \mathbb{R}^n , $n = 4, 8, 10, 2$ constraints.

Objective function:

$$\max f(\mathbf{x}) = \sum_{i=1}^n [10 \cos(2\pi x_i) - x_i^2] - 10n$$

Constraints:

$$\begin{aligned} \sum_{i=1}^n x_i^2 - 1 &= 0 \\ \sum_{i=1}^n x_i - n &\leq 0 \end{aligned}$$

Variable bounds:

$$x_i = [-3.12, 5.12], \quad i = 1, \dots, n$$

REFERENCES

- [1] E. Hansen, *Global Optimization Using Interval Analysis*. New York: Marcel Dekker, 1992.
- [2] H. Ratschek and J. Rokne, "Interval methods," in *Handbook of Global Optimization*, R. Horst and P. M. Pardalos, Eds. Dordrecht: KAP, 1995, pp. 751–828.
- [3] S. Skelboe, "Computation of rational interval functions," *BIT*, vol. 14, pp. 87–95, 1974.
- [4] E. Hansen and S. Sengupta, "Global constrained optimization using interval analysis," in *Interval Mathematics*, K. Nickel, Ed. Berlin: Springer-Verlag, 1980, pp. 25–47.
- [5] R. E. Moore, *Interval Analysis*. Englewood Cliffs: Prentice Hall, 1966.
- [6] D. Ratz and T. Csendes, "On the selection of subdivision directions in interval branch-and-bound methods for global optimization," *J. of Glob. Opt.*, vol. 7, pp. 183–207, 1995.
- [7] L. G. Casado, I. García, and T. Csendes, "A heuristic rejection criterion in interval global optimization algorithms," *BIT*, vol. 41, no. 4, pp. 683–692, 2001.
- [8] L. G. Casado, J. A. Martínez, and I. García, "Experiments with a new selection criterion in a fast interval optimization algorithm," *J. of Glob. Opt.*, vol. 19, pp. 247–264, 2001.
- [9] T. Csendes, "New subinterval selection criteria for interval global optimization," *J. of Glob. Opt.*, vol. 19, pp. 307–327, 2001.
- [10] D. B. Skillicorn and D. Talia, "Models and languages for parallel computation," *ACM Computing Surveys*, vol. 30, no. 2, pp. 123–169, 1998.
- [11] T. Henriksen and K. Madsen, "Use of a depth-first strategy in parallel global optimization," Institute of Numerical Analysis, Technical University of Denmark, Lyngby, Tech. Rep. 92-10, 1992.
- [12] J. Eriksson and P. Lindstrom, "A parallel interval method implementation for global optimization using dynamic load balancing," *Reliable Computing*, vol. 1, no. 1, pp. 77–92, 1995.
- [13] A. Benyoub and E. M. Daoudi, "Parallelization of the continuous global optimization problem with inequality constraints by using interval arithmetic," in *Proceedings of the 9th International Conference on High-Performance Computing and Networking*. Springer-Verlag, 2001, pp. 595–602.
- [14] S. Ibraev, "A new parallel method for verified global optimization," PhD Thesis, University of Wuppertal, Germany, 2001.
- [15] C.-Y. Gau and M. A. Stadtherr, "Parallel interval-Newton using message passing: dynamic load balancing strategies," in *Supercomputing '01: Proceedings of the 2001 ACM/IEEE Conference on Supercomputing (CDROM)*. New York, NY, USA: ACM Press, 2001, pp. 23–23.
- [16] S. Berner, "Parallel methods for verified global optimization: practice and theory," *J. of Glob. Opt.*, vol. 9, pp. 1–22, 1996.
- [17] Y. Wu and A. Kumar, "Interval subdivision strategies for constrained optimization," in *International Conference on Numerical Analysis and Applied Mathematics*, Greece, 2005, pp. 593–596.
- [18] C. A. Floudas, P. M. Pardalos, C. S. Adjiman, W. R. Esposito, Z. H. Günius, S. T. Harding, J. L. Klepeis, C. A. Meyer, and C. A. Schweiger, *Handbook of Test Problems in Local and Global Optimization*. Dordrecht: KAP, 1999.
- [19] CUTeR, "CUTeR: A constrained and unconstrained testing environment, revisited." <http://cuter.rl.ac.uk/cute-www/problems.html>.
- [20] Message Passing Interface Forum, "MPI: The message passing interface standard," 1994.
- [21] A. Törn and A. Žilinskas, *Global Optimization, Vol. 350 of Lecture Notes in Computer Science*. Berlin: Springer-Verlag, 1989.