Journal of Global Optimization **9**: 1–22, 1996.

Parallel Methods for Verified Global Optimization Practice and Theory

Sonja Berner
Fachbereich Mathematik, Universität Wuppertal, D-42097 Wuppertal
email: sonja@math.uni-wuppertal.de

Abstract. We present a new parallel method for verified global optimization, using a *centralized mediator* for the dynamic load balancing. The new approach combines the advantages of two previous models, the master slave model and the processor farm. Numerical results show the efficiency of this new method. For a large number of problems at least linear speedup is reached. The efficiency of this new method is also confirmed by a comparison with other parallel methods for verified global optimization. A theoretical study proves that using the best-first strategy to choose the next box for subdivision, no real superlinear speedup may be expected concerning the number of iterations. Moreover, the potential of parallelization of methods of verified global optimization is discussed in general.

Key words: Global optimization, parallel computing, interval arithmetic, branch and bound, dynamic load balancing

1. Introduction

In this paper, which contains results from Berner (1995) we are dealing with the parallelization of verified methods for global optimization. The problem considered here can be formulated as: Given a continuously differentiable function $f : D \to \mathbb{R}$, $D \subseteq \mathbb{R}^n$ open, and a compact set $X^0 \subseteq D$ which shall be an n-dimensional box, find the global minimum

$$f^* = \min_{x \in X^0} f(x)$$

and also find the set of all global minimum points

$$X^* = \{x \in X^0 \ : \ f(x) = f^*\}.$$

Classical methods may fail to find the global minimum since often only finitely many discrete points are considered. Interval methods, however, find an *enclosure* of the global minimum and also of all global minimum points with the *branch and bound* principle. According to this principle the starting box X^0 is successively subdivided into smaller subboxes, and those which can be shown to not contain global minimum points are deleted.

Problems of global optimization are usually hard to solve. Thus the development of parallel methods often is a must to make them tractable

at all. After some preparations in Sections 2 and 3 we present a new approach to the parallelization of branch and bound methods for verified global optimization in Section 4. This approach uses a *centralized mediator* for dynamic load balancing. The efficiency of this new method is shown by measurements on a CM5 parallel computer for a variety of test problems. It is compared with other methods, which are briefly described (Section 5).

Some theoretical results given in Section 6 show that an efficient parallelization may especially be expected for those problems with a high number of global minimum points or many local minimum points, with function value near to f^*. These problems are hard to solve by classical methods. Moreover, we prove that no 'true' superlinear speedup may be expected when the best-first strategy is used to choose the next box for subdivision.

2. Interval Arithmetic

The use of interval arithmetic (see e.g. Alefeld and Herzberger (1983), Neumaier (1990)) allows computations with intervals instead of real numbers. Let the set of all compact intervals be denoted by $\mathbb{IR} := \{[\underline{a}, \overline{a}] : \underline{a}, \overline{a} \in \mathbb{R}, \underline{a} \leq \overline{a}\}$. The operations $+, -, \cdot, /$ for intervals $A, B \in \mathbb{IR}$ are defined by

$$A \circ B := \{a \circ b : a \in A, b \in B\}, \quad \circ \in \{+, -, \cdot, /\},$$

where $0 \in B$ is excluded for division. To compute $A \circ B$, only the bounds of A and B have to be considered. Standard functions for intervals can be defined in an analogous way. The bounds of an interval $A = [\underline{a}, \overline{a}] \in \mathbb{IR}$ are termed $\inf A := \underline{a}$ and $\sup A := \overline{a}$ here. The diameter of A is denoted by $w(A) := \overline{a} - \underline{a}$, the absolute value by $|A| := \max\{|a| : a \in A\}$. Interval vectors $X \in \mathbb{IR}^n$ are also termed *boxes* here.

If interval arithmetic is used on a machine, then, additionally, outward rounding is necessary to guarantee that $a \circ b$ is enclosed in the machine interval $A \circ B$ for all $a \in A$, $b \in B$. There exist several programming languages, such as Pascal-XSC (Klatte *et al.* (1991)), C-XSC (Klatte *et al.* (1993)) etc., and also libraries which support outward rounding.

3. A Basic Serial Method

Methods of verified global optimization do not only use information at discrete points. Instead, they use global information in the form of an

inclusion function F to find the solution of the problem. This inclusion function F provides an interval $F(X)$ with

$$F(X) \supseteq \{f(x) : x \in X\} \quad \text{for all } X \subseteq X^0,\ X \in \mathbb{IR}^n,$$

which encloses the range of f over X.

Moreover, the *branch and bound* principle is applied. The starting box X^0 is successively subdivided into smaller subboxes, and boxes X for which we can decide that they do not contain global minimum points are deleted. This decision is made by use of the inclusion function F and an upper bound $\tilde{f}$ of the global minimum. Boxes X with $\inf F(X) > \tilde{f}$ reliably do not contain any global minimum point and, therefore, can be deleted. An introduction to these methods can be found in Hansen (1992), Ratschek and Rokne (1988).

Below, a basic serial method for verified global optimization is described. Boxes are stored in a working list L or in a solution list $\tilde{L}$ together with $\inf F(X)$, the lower bound of the range.

ALGORITHM 1 *Basic serial method for verified global optimization*

Given: starting box X^0, inclusion function F, upper bound $\tilde{f}$, $\varepsilon > 0$

initialize $L := \{(X^0, \inf F(X^0))\}$, $\tilde{L} := \{\}$;
while $L \neq \{\}$ **do**
  take pair (X, x) out of list L;
  divide X into subboxes $X^1, \ldots, X^k$, $k \geq 2$;
  for $i = 1, \ldots, k$ **do**
    compute $F(X^i)$;
    if $\inf F(X^i) \leq \tilde{f}$ **then**
      $\tilde{f} := \min\{\sup F(X^i), \tilde{f}\}$;
      if $w(F(X^i)) \leq \varepsilon$ **then** put $(X^i, \inf F(X^i))$ into $\tilde{L}$;
                else put $(X^i, \inf F(X^i))$ into L;
    end if;
  end for;
  cut-off test: delete all (X, x) with $x > \tilde{f}$;
end while;

To get a full description of the algorithm one has to decide, how to choose the next box for subdivision, and how this subdivision works.

Concerning the choice of the next box, several strategies are known in literature:

– *oldest-first* strategy,

– *depth-first* strategy,

– *best-first* strategy.

The oldest-first strategy always chooses the pair (X, x) for subdivision, having resided in the list L longest. The depth-first strategy always takes one of the pairs last inserted into L. If the best-first strategy is applied, the pair (X, x) with x minimal in L is chosen.

One can show the following theorem (Berner (1995), Berner (1996c)):

THEOREM 1. Using Algorithm 1 with best-first strategy no pair (X, x) with $x > f^* + \varepsilon$ will be chosen for subdivision.

Here, ε is the parameter which was chosen for the criterion to insert boxes into the solution list $\tilde{L}$. It is important to note that any strategy for choosing the next box will have to investigate pairs (X, x) with $x \leq f^*$ and X not fulfilling the termination criterion $w(F(X)) \leq \varepsilon$. For these boxes a decision, if they contain global minimum points or not is not possible yet, even if the global minimum f^* is already known. Only pairs (X, x) with $f^* < x \leq f^* + \varepsilon$, therefore, might be considered unnecessarily with the best-first strategy, but the number of these pairs is small in practice. Using the other two strategies, normally many pairs (X, x) with $x > f^* + \varepsilon$ are investigated, so that it is favorable to use the best-first strategy. This was always our choice in our algorithms.

Subdivision of boxes is possible e.g. by multisection (Berner (1995), Berner (1996c)): a box is bisected once or several times whereby the directions basically are chosen as those where a merit function $D_i(x)$ is maximal. This merit function may be defined by (cf. Csendes and Ratz, Ratz and Csendes (1995))

−strategy A: $D_i(X) := w(X_i)$,

−strategy B: $D_i(X) := w(F_i'(X)) \cdot w(X_i)$,

−strategy C: $D_i(X) := |F_i'(X)| \cdot w(X_i)$.

The usual way in literature is to bisect a box only once by use of strategy A. In Berner (1995), Berner (1996c) it was shown that especially for large problems it is favorable to apply strategy C and that always better results were obtained by bisecting a box twice in each step, i.e. subdividing it into four new subboxes. The two directions of bisection are chosen in the following manner. The first direction i is chosen as the one with $D_i(X)$ largest. Then we set $D_i(X) := D_i(X)/2$ and choose the second direction for bisection again as the direction with largest $D_i(X)$.

In Berner (1996c) it was shown that in this way a very efficient serial method has been constructed. This is quite important to keep in mind since the efficiency of our parallel method is always measured with respect to the efficiency of this serial method.

4. Parallelization

The simplest way to parallelize an algorithm like the one presented as Algorithm 1 is to partition the starting box X^0 to all processors. Then each processor can work on its own subbox rather independently of all others. But it is likely that the amount of work one has to spend on a subbox to solve the global optimization problem there, varies for different subboxes. This approach, therefore, is not sufficient. Some processors may become idle prematurely. This implies the need of *dynamic load balancing.*

Moreover, one has to pay attention to that not considerably more boxes are investigated in the parallel case than in the serial one. This can happen, since in the parallel algorithm the best-first strategy is only applied locally on each processor. From the global point of view, boxes may be investigated in a different order than in the serial method. This implies that there should exist an exchange of the current best value for the upper bound $\tilde{f}$ between the processors. A better value for $\tilde{f}$ should be sent as fast as possible to all other processors, but also without a large overhead in communication. Furthermore, one should try to work always on the p "best" pairs, i.e. those (X, x) with smallest lower bound x, when p is the number of processors used.

In the following we will first discuss some existing parallelizations, and then present our new parallel approach.

4.1. Existing Parallelizations

The approach of Henriksen and Madsen (1992) uses a master-slave model. All boxes are kept in a central list on the master; the same is true for the current upper bound $\tilde{f}$. The master sends boxes to the slaves and gets the results back. A disadvantage of this model is that there is quite a lot of work for the master. It becomes a bottleneck. Moreover, the length of the list L is limited by the memory of the master, while the memory of the slaves is not used at all. Henriksen and Madsen applied the depth-first strategy to minimize communication. A slave always keeps one of the two new subboxes resulting from the last subdivision for further investigation, and only sends the second one to the master. But applying the depth-first strategy, a good upper bound $\tilde{f}$ has to be known in advance to get an efficient algorithm (cf. Section 3). In their approach, Henriksen and Madsen used the global minimum f^* to initialize $\tilde{f}$, but this value is normally unknown when the algorithm starts.

Eriksson (1991) used a processor farm for his parallel method. The processors are organized in a ring, each of them works in the same way.

They keep their own sorted list, the best-first strategy is applied on each processor. If some processor runs out of boxes, it sends a request to the next neighbor in the ring. Requests are forwarded by processors which are idle, too. A better value for the current upper bound $\tilde{f}$ is sent by a broadcast to all others. Two processes were used on each processor: a worker to investigate the subboxes and a scheduler responsible for the dynamic load balancing. An approach like this could not have been realized on the machine we used here, a Connection Machine CM5, since only one process may run on one processor at the same time.

The approach of Moore, Hansen and Leclerc (1992) is similar to the one of Eriksson. They also use a processor farm, but the processors are not organized in a ring. A request, therefore, is sent to an arbitrary processor. If this processor cannot respond to this request, it sends a 'no' back, and another processor is tried. The main difference to the approach of Eriksson is that here the oldest-first strategy was used. A speedup of 170 on 32 processors was reached, but this is largely due to the fact that the serial method is inefficient.

In fact, in a later paper Leclerc (1993) used nearly the same algorithm, but applied the best-first strategy now. He gained a factor of 78 in speed for the serial method for the problem considered in Moore *et al.* (1992). But for the parallel method now only about half of linear speedup was reached.

4.2. A New Parallel Approach

Our new parallel approach combines the principle of a master-slave model with that of a processor farm. Each processor keeps its own sorted list; the best-first strategy is applied on each processor. Unlike the processor farm one processor is determined to be the *centralized mediator*. This processor does not work on boxes. Instead, it waits for requests of idle processors to send them new boxes. Moreover, it keeps a limit *max*, that is changed dynamically. This limit is used to make sure that the centralized mediator does not run out of boxes, but also that not too many boxes are stored in its list. Processors which keep more than *max* boxes in their list send some of them to the centralized mediator (Fig. 4.2).

A similar approach was used in Smith and Schnabel (1992) to parallelize a classical method for global optimization, the multi level single linkage method (cf. Rinnoy Kan and Timmer (1987)). The term "centralized mediator" has been taken over from that paper. In Smith and Schnabel (1992), other communication models have also been examined, but numerical experiments indicated that the model using a centralized mediator was the most efficient one.

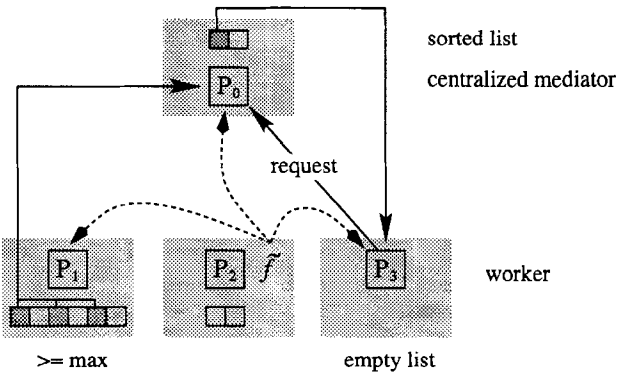

Figure 1. Communication structure used with centralized mediator.

An advantage of the parallel approach presented here compared to the master-slave model used in Henriksen and Madsen (1992) is that there is less work for the centralized mediator. So it will not become a bottleneck supposed that the number of processors used is not too high. For the master-slave model in Henriksen and Madsen (1992), however, the master already became a bottleneck for only 32 processors. Moreover, the whole memory including that of the workers is used.

Compared to the approach of Eriksson and the one of Moore, Hansen and Leclerc, there is no need to request several processors to get boxes, if a processor becomes idle. Instead, it is the centralized mediator that directly responds to each request.

Now, we will present our parallel method in some more detail.

ALGORITHM 2 *Parallel algorithm for the worker*

initial phase, provides L, $\tilde{L}$, $\tilde{f}$;
repeat
  while $L \neq \{\}$ **do**
    take (X, x) with x minimal out of L and subdivide X;
    (cf. serial method)
    possibly receive better value for $\tilde{f}$;
    send own value of $\tilde{f}$, if this is minimal;
    cut-off test;
    possibly send boxes to the centralized mediator for
    dynamic load balancing;
  end while;
  send request to the centralized mediator;
  receive boxes from the centralized mediator;
  if number of received boxes > 0 **then** sort them into L;
                else stop;
end repeat;

The algorithm starts with an initial phase, which is common for the workers and the centralized mediator (Algorithm 2 and 3). The starting box is partitioned, and each processor gets one subbox. The upper bound $\tilde{f}$ is initialized, e.g. with $\tilde{f} := \min_{i=1}^{p} \sup F(X^i)$. Here X^i is the starting subbox of processor i, and p is the number of processors used. A loop follows, during which a worker is first investigating the boxes of its own list until it is empty. Then it sends a request to the centralized mediator and waits for a response. The number of boxes sent by the centralized mediator might be zero to signal termination. The algorithm will stop then. Otherwise the boxes are sorted into the list L, and the worker will restart investigation of the boxes of its list. A box of the list L is investigated in the same way as in the serial method, i.e. it is subdivided and the new subboxes are inserted into L, into $\tilde{L}$, or they are deleted.

After one box has been investigated, possibly better, i.e. lower, values for the upper bound $\tilde{f}$, determined by other processors, are received. The value of $\tilde{f}$ is only updated if the received value is lower than the current one. If the current value of $\tilde{f}$ found by the worker itself is smaller than the value in the last iteration and also than all received values, then the worker will send this value to all others.

For the reason of dynamic load balancing, the value of *max* might be updated if a new message with a value of *max* of the centralized mediator is waiting to be received. If the list L of the processor actually contains more than *max* elements, then it will send some of its boxes to the centralized mediator. In this case, the second, the fourth, and so on pair of the list L is chosen to be sent, because not only "bad" pairs (X, x) with high lower bound x shall be sent, but the processor itself shall also keep some "good" pairs with small lower bound x.

The algorithm of the centralized mediator (Algorithm 3) also starts with the initial phase. Then a loop is executed until $p - 1$ processors will wait for boxes. This means that these processors and also the centralized mediator itself do not have any boxes in their list L. For termination, the centralized mediator then sends a termination message corresponding to sending zero boxes to all processors.

ALGORITHM 3 *Parallel algorithm for the centralized mediator*

initial phase, provides L, $\tilde{L}$, $\tilde{f}$;
while not all $p - 1$ workers are waiting for boxes **do**
  receive requests, boxes and possibly a better value for $\tilde{f}$;
  respond to requests;
  cut-off test;
  adapt value of *max*;

end while;
send message of termination (send zero boxes);

While executing the loop, the dynamic load balancing is carried out. Requests are received, and also boxes from those processors which have plenty of them. These boxes are passed to those processors which have sent a request.

Moreover, the centralized mediator possibly also receives a better value for the upper bound $\tilde{f}$. It is important to know the current best upper bound $\tilde{f}$ of all processors. Otherwise, boxes might be sent to other processors which could already be deleted knowing the current best upper bound. The cut-off test is used afterwards to get rid of those pairs (X, x) in L with $x > \tilde{f}$, since these should not be sent to other processors any more. So after the cut-off test the length of list L gives the number of boxes for which it is actually reasonable to be sent to other processors.

Furthermore, the value of *max* is adapted depending on the actual length of the list L of the centralized mediator and also on the number of requests that could not have been answered until now. The value of *max* is sent to all other processors using asynchronous communication. Here we take advantage of an asynchronous communication routine of the Connection Machine CM5 that allows changing the message as long as it is not already sent. Thus, the actual value is always received. One can gain a number of messages by just changing the value of *max*, if the message has not been sent yet. The same holds for the communication of a better value of the upper bound $\tilde{f}$.

The algorithms are only shown in their simplest version here. More sophisticated details are given in the next section. The presentation of the algorithms, anyway, is also simplified.

4.3. Improvements

A more sophisticated initial phase than the one explained above can be used. It is given in Algorithm 4. After having partitioned the starting box each processor starts a classical local optimization from the midpoint of its subbox. This usually delivers a rather good value for the upper bound $\tilde{f}$. Some of the subboxes may already be deleted. For this reason, a synchronous load balancing has to follow, which provides a good distribution of subboxes.

ALGORITHM 4 *Initial phase (for processor me)*

get own subbox X^{me} from partition of the starting box X^0;
start local optimization from the midpoint of X^{me} providing $\hat{x}^{me} \in \mathbb{R}^n$;

$\tilde{f} := \min_{i=1}^{p} \sup F(\hat{x}^i)$;
if $\inf F(X^{me}) > \tilde{f}$ **then** delete X^{me};
repeat
carry out synchronous load balancing by subdividing and redistributing the remaining boxes;
compute $F(X^i)$ for own subboxes X^i, $i = 1, \ldots, k$, and delete those X^i with $F(X^i) > \tilde{f}$;
$a :=$ number of processors which keep at least one subbox;
until $(a \geq p/2) \vee (a \leq p/8)$;
$L := \{\}$; $\tilde{L} := \{\}$;
for all subboxes X^i of processor me **do**
sort $(X^i, \inf F(X^i))$ into L or put it into $\tilde{L}$;

This new initial phase is quite important for problems with high running time, but which do not parallelize very well. For problem MHL, e.g., about half of the execution time is saved when this more sophisticated initial phase is applied. For some small problems with low running time the algorithm may be less efficient since the time used by the local optimization and the synchronous load balancing is not compensated by a better starting value for $\tilde{f}$.

To avoid considering boxes in the parallel method which are not considered in the serial case when the best-first strategy is applied to the whole list of all boxes, we also carry out a load balancing concerning the *quality* of pairs in the list L. With regard to the best-first strategy, high quality of a pair (X, x) means that the lower bound x is small. A load balancing concerning the quality of pairs is also applied in Eriksson (1991) and in Leclerc (1993). To keep communication low, the smallest lower bound x of all pairs in the list L of a processor, characterizing its pair with highest quality, is only sent together with a better value for the upper bound $\tilde{f}$. This is sufficient, since it is very likely for processors with pairs of high quality to find a better value for $\tilde{f}$. After the lower bound x together with $\tilde{f}$ has been sent to all other processors, they compare the quality of their own pairs with the received lower bound x. A processor may send a request to the corresponding processor, if only pairs of sufficiently low quality are available in its own list.

It may happen that not enough boxes are available in the list of the centralized mediator to answer to all current requests. Since the effort which has to be spent to completely investigate one box is not known in advance, it might sometimes be better to subdivide some boxes in smaller parts instead of letting several processors wait. In these situations it turned out that it was normally better to split boxes in order to be able to answer to all requests instead of always sending a whole box to each processor and letting some processors wait if not

enough boxes are available. For example, for problem MHL we obtained an improvement of about 37%.

5. Numerical Results

The efficiency of a parallel algorithm may be measured by the speedup

$$S(p) := \frac{\text{execution time of the method using 1 processor}}{\text{execution time of the parallel method using } p \text{ processors}}$$

of the parallel method compared to the serial one. To be more honest, sometimes the execution time of the *best* serial algorithm is used to compute the speedup, but often this best algorithm is not known.

Our parallel method was implemented in Pascal-XSC (Klatte *et al.* (1991)) on a Connection Machine CM5 with 32 nodes. To this purpose the programming language was ported to the CM5 (Berner (1995)). The CM5 is an MIMD-machine using message passing for communication. The processors are 40 MHz SPARC processors, each with a local memory of 32 MByte. The topology of the communication network is a *fat-tree*, which means that each processor can communicate with each other in nearly the same time.

The numerical results presented in this section were obtained with the basic parallel algorithm given by Algorithm 2 and 3 with all improvements of Section 4.3. Additionally, the monotonicity test and, for small boxes, the non-convexity check and the interval Newton method (cf. Hansen (1992), Ratschek and Rokne (1988)) were applied. Unless otherwise stated, strategy C is used for subdivision and a box is always bisected twice in each step.

The test problems considered are partly standard test functions. Only test functions for which parallelization might be useful were considered. Problems which only need up to 100 iterations to find the solution have not been considered, since a parallelization on 32 processors is not promising at all. Some standard test problems have been taken from Törn and Žilinskas (1989); these are the Hartman function H6 and the problem of Goldstein-Price GP. The Levy function LEVY3, the Kowalik problem KOW and the problems Schwefel 2.7 (SC2.7) and Schwefel 2.14 (SC2.14) are described in the book of Hansen (1992) and in Walster *et al.* (1985). Here $[0, 0.42]^4$ was chosen as starting box for problem KOW, $[0, 5] \times [8, 11] \times [0.5, 3]$ for problem SC2.7, all others like those given in Hansen (1992), Walster *et al.* (1985). Some parameter estimation problems were also used as test problems. These are the problem MHL described in Moore *et al.* (1992), the problem CSEN (cf. Csendes and Ratz) and the problems GEO1 to GEO3 arising from

geodesy. These problems, together with problems HM1 to HM6 used in Henriksen and Madsen (1992) and the problems UEND1 and UEND2, for which X^* is infinite, are reproduced in the appendix. For problem CSEN the parameter $\varepsilon = 10^{-3}$ was used for insertion into the solution list, for all other problems $\varepsilon = 10^{-6}$ was chosen.

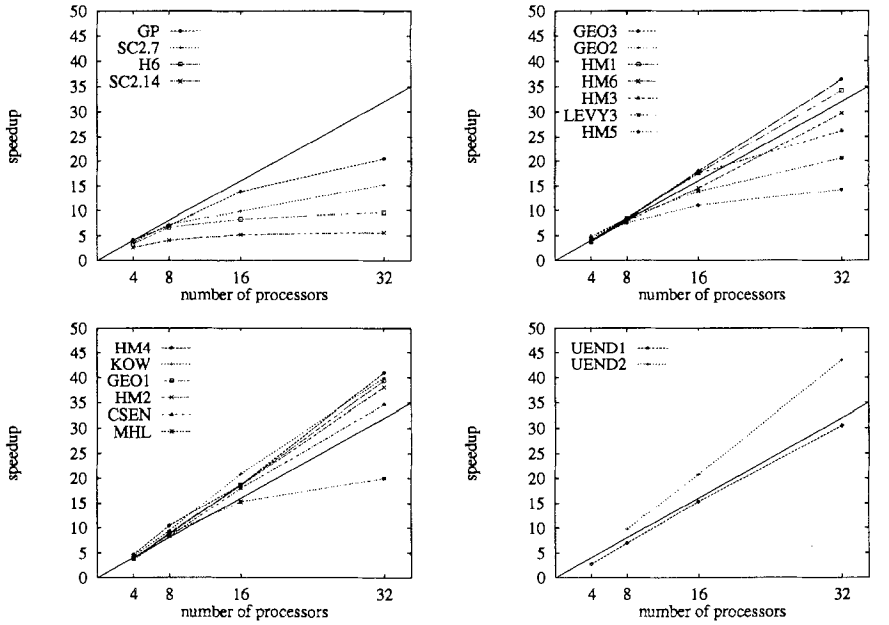

Figure 2. Speedup obtained for small problems with serial running time less than 100 s (top left), for medium problems with running time from 100 s up to 1000 s (top right), for large problems with running time from 1000 s up to 17000 s (bottom left) and for problems with infinitely many global minimum points (bottom right).

Figure 5 shows the speedup for different test problems reached by the parallel method using 4, 8, 16 and 32 processors. In Figure 5 on the top left the speedup for rather small problems with low running time (less than 100 s) is given. One sees that no linear speedup is reached, for two of the problems even less than half of linear speedup is obtained. But as the running time increases (100 s to 1000 s, top right) the speedup gets better. It is always higher than half of linear speedup, and for some problems it is even superlinear. For problems with high running time from 1000 s to 17000 s with one exception in all cases superlinear speedup is reached (bottom left). The picture on the bottom right shows the speedup for two problems with infinitely many global minimum points, which is at least about linear. This kind of problem parallelizes very well. This is also pointed out in Section 6.1.

It is difficult to compare the efficiency of parallel methods only by a comparison of speedups. However, since different computer architec-

tures were used in the different approaches, this was the only possible way here.

For the following comparison it is worth mentioning that, using strategy C instead of A, for some problems the serial as well as the parallel method becomes faster but the speedup decreases. The explanation for this is a higher improvement in speed for the serial than for the parallel method when switching from strategy A to C. For example, for problem HM5 a rather high speedup of 33.5 is reached on 32 processors with strategy A, the parallel method takes 42.1 s for termination. Using strategy C the speedup decreases to 14.2 (cf. Fig. 5), although the parallel method only needs 10.5 s (instead of 42.1 s with

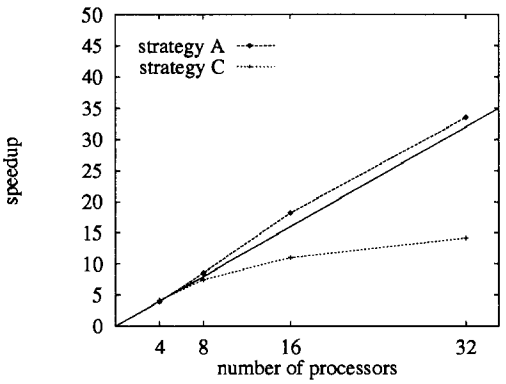

Figure 3. Sometimes speedup decreases when strategy C is used instead of A (here for problem HM5).

strategy A). The same is true for problem MHL, the only one of the larger problems for which no linear speedup is reached. The speedup, therefore, has to be regarded with caution. It depends highly on the efficiency of the serial method used as the base for its computation.

For the following comparison (by speedups) we assume the serial method used here to be as efficient as those used for other parallelizations (cf. Section 3). Often it definitely will be more efficient due to a better subdivision strategy, since all other methods use strategy A to subdivide boxes.

In Table 5 the speedup of the new parallel method presented here is compared to the speedup obtained by Henriksen and Madsen (1992). For their method, the higher speedup is presented, which was reached by use of the depth-first strategy and with the initialization $\tilde{f} = f^*$. With one exception (problem HM5) higher speedup is always reached with our new parallel method. If we use strategy A for this problem as was the case in Henriksen and Madsen (1992), then our speedup is much higher than theirs in this case (values given in brackets).

Table I. Comparison with the parallel method of Henriksen and Madsen.

problem	speedup 16 processors Hen./Mad.	new method		speedup 32 processors Hen./Mad.	new method	
HM1	8.6	17.6		7.3	34.2	
HM2	10.4	18.6		12.6	38.1	
HM3	11.4	17.5		11.7	26.2	
HM4	13.5	18.7		16.5	41.0	
HM5	13.0	11.0	(18.2)	20.7	14.2	(33.5)
HM6	11.6	14.5		19.2	29.7	

A comparison with the parallel method of Eriksson is given in Table 5. Since three different modifications of a parallel method are considered in Eriksson (1991), the highest speedup obtained there is always used for comparison. For problem HM6 we got a higher speedup with our new

Table II. Comparison with the parallel method of Eriksson.

problem	speedup 16 processors Eriksson	new method		speedup 32 processors Eriksson	new method	
SC2.14	19.6	5.2	(6.3)	28.3	5.6	(7.3)
SC2.7	11.0	9.8	(18.1)	15.0	15.2	(34.8)
HM6	12.0	14.5		30.9	29.7	

parallel approach on 16 processors; on 32 processors the speedup was slightly lower. For problem SC2.7 our speedup becomes higher than the one reached by Eriksson, if strategy A is used instead of C. For problem SC2.14 the speedup remains worse, even if strategy A is used. But this problem needs about 10 times the running time of problem SC2.7 with the serial method of Eriksson, while only about 6% of the time used to solve problem SC2.7 is needed with our serial method. So the high speedup reached by Eriksson might result from an inefficient serial method in this case.

For problem MHL considered in Moore *et al.* (1992) a speedup of 170 on 32 processors, as reported in Moore *et al.* (1992) is not reached by our method. But the speedup of 20.0 and 40.6, respectively, with strategy A on 32 processors is still higher than the speedup of about 8.5 on 20 processors reached by Leclerc (1993) for this problem.

The comparison shows that for many problems our speedup was higher than the one of other parallel methods for global optimization. Moreover, one has to keep in mind that all other parallel methods use strategy A for subdivision while, here the more efficient strategy C was used, which sometimes causes the speedup to decrease, although the total execution time of the parallel method is decreased, too. If we also use strategy A for these problems, then there remain two problems for which the speedup given in Eriksson (1991) is higher. But for one of these problems, the serial method of Eriksson seems to be inefficient, for the other one the speedup is only slightly lower on 32 processors. Furthermore, the approach of Eriksson uses two processes on each processor, which could not be realized on the Connection Machine CM5. Since our serial method can be supposed to be at least as efficient as the one used to compute the speedup for the other parallel methods, the comparison shows that a fairly efficient parallel method has been constructed here.

6. Some Theoretical Results

6.1. General Considerations

The parallel algorithm differs somewhat from the serial one, since the best-first strategy is only applied locally on each processor. The question arises, therefore, if the same results will be obtained by both algorithms. One can show that this is true under certain assumptions (Berner (1995)):

THEOREM 2.Let the serial algorithm be started under the same conditions that were obtained for the parallel method after the initial phase, i.e. let $L_{ser} = \cup_{i=1}^{p} L_{par}^{i}$, $\tilde{L}_{ser} = \cup_{i=1}^{p} \tilde{L}_{par}^{i}$ *and* $\tilde{f}_{ser} = \tilde{f}_{par}$ *at the beginning. Here* L_{par}^{i} *and* $\tilde{L}_{par}^{i}$ *are the lists of processor* i, $i = 1, \ldots, p$, *after the initial phase,* p *is the number of processors used.*

This implies that after termination of the serial and of the parallel algorithm $\tilde{f}_{ser} = \tilde{f}_{par}$ *and* $\tilde{L}_{ser} = \cup_{i=1}^{p} \tilde{L}_{par}^{i}$ *holds, i.e. the same results are obtained.*

The proof is rather technical, and will be omitted here. It can be found in Berner (1995).

Problems of global optimization might parallelize poorly, if there exists only one global minimum point and if subboxes which do not contain the global minimum point can directly be deleted, e.g. by using an inclusion function that is equal to the range of the function.

But one can show that if there exist at least p global minimum points and if those will be separated by a low number of subdivisions, then a good parallelization is possible, since enough boxes are available to keep all processors working. This is also true if at least p points exist with a function value only slightly higher than the global minimum. Note that problems of this kind are usually hard to solve by classical methods.

Assume that a box is always split into q subboxes using multisection. Let $X^{(0,j)}$, $j = 1, \ldots, p$, be the boxes arising from the first partition of the starting box to all p processors in the initial phase. Let $X^{(1,j)}$, $j = 1, \ldots, k_1$, $k_1 = p \cdot q$, be those boxes arising from the first multisection of all $X^{(0,j)}$, and so on. Let the number of those $X^{(i,j)}$, $i \in \mathbb{N}_0$, which contain global minimum points be denoted by $a(i)$:

$$a(i) := \#\{X^{(i,j)},\, j = 1, \ldots, k_i \,:\, X^{(i,j)} \cap X^* \neq \emptyset\}.$$

Let $\varepsilon = 0$ and assume that all processors work synchronously on their boxes. Let the boxes be redistributed after each step to keep as many processors working as possible. If only $k < p$ global minimum points exist, then it is possible that after some time only k boxes will be investigated any longer, each of them containing a global minimum point. This means that some processors might become idle. Assuming, however, that there exists an $m \in \mathbb{N}_0$ with $a(m) \geq p$, and denoting the smallest such m by m_0, we get:

LEMMA 1. Let $m_0 := \min\{m \in \mathbb{N}_0 : a(m) \geq p\}$, whereby $m \in \mathbb{N}_0$ with $a(m) \geq p$ should exist. Assuming that $\varepsilon = 0$ is chosen in the parallel algorithm, it holds that the number of steps where less than p boxes are available is at most m_0.

Proof. If some processor P_1 is running out of boxes during the iteration, then two cases can occur:

1. Some other processor keeps more than one box in its list L. Then one of them can be passed to P_1.

2. Each processor keeps at most one box in its list. Using the assumption and $a(m) \geq p$ for all $m \geq m_o$, one knows that at least one processor exists with more than one global minimum point in its box. After at most m_0 iterations this processor will keep two boxes in its list. One of them can be sent to P_1.

Let $\tilde{m}$ be the number of iterations P_1 is idle. Since each processor will subdivide its only box during these $\tilde{m}$ iterations, only boxes $X^{(m,i)}$ with $m \geq \tilde{m}$ will exist afterwards. Keeping this in mind, it is obvious that

the number of iterations with less than p boxes available is at most m_0. □

It follows

COROLLARY 1. Let there exists an $m \in \mathbb{N}_0$ with $a(m) \geq p$, then there exists a point of time from which all p processors can be kept busy.

This means that if there are at least p global minimum points and if these are well distributed over the starting box X^0 then the method will parallelize quite nicely.

6.2. Superlinear Speedup?

The numerical results of Section 5 show that, for some problems slightly superlinear speedup is reached. Superlinear speedup often creates suspicion that the serial algorithm used for comparison is inefficient. Let us assume in the following that the best-first strategy is used to choose the next box for subdivision. According to Theorem 1 almost only those boxes are considered for which this is really necessary. Therefore, the question arises, whether superlinear speedup is possible with respect to the number of iterations, i.e. the number of investigated boxes. The next theorem shows that this may hardly be expected.

THEOREM 3. Assume that the serial Algorithm 1 with best-first strategy is started under the same conditions that occur in parallel after the initial phase, i.e. let the serial algorithm start with working list $L_{ser} = \bigcup_{i=1}^{p} L_{par}^i$, where L_{par}^i denotes the working list of processor i resulting from the initial phase.

Let $I(1)$ denote the number of iterations of the serial method, K the number of boxes X investigated in serial with $f^* < \inf F(X) (\leq f^* + \varepsilon)$. Assume that, applying the parallel method, the maximal number of iterations on one processor is $I(p)$. Then

$$\frac{I(1)}{I(p)} \leq \frac{p}{1 - \frac{K}{I(1)}}$$

holds.

Proof. Theorem 1 shows that, using the best-first strategy no pair (X, x) is investigated for which $x > f^* + \varepsilon$ holds. With K defined as above, the number of investigated pairs (X, x) with $x \leq f^*$ is $I(1) - K$. These boxes X also have to be investigated in the parallel method, since a decision whether a global minimum point is contained in the box X is not possible yet, even if the global minimum f^* is already known.

Therefore, there exists at least one processor which subdivides $\left\lceil \frac{I(1)-K}{p} \right\rceil$ or more boxes, such that at least $\left\lceil \frac{I(1)-K}{p} \right\rceil$ iterations are necessary in the parallel case. This implies

$$I(p) \geq \frac{I(1)-K}{p} = \frac{I(1)}{p}\left(1 - \frac{K}{I(1)}\right) \quad \text{resp.} \quad \frac{I(1)}{I(p)} \leq \frac{p}{1-\frac{K}{I(1)}}.$$

□

As we already mentioned in Section 3 the number K normally is small compared to the total number of iterations $I(1)$. Therefore the speedup on p processors is limited by a value only slightly higher than p.

This theorem only deals with the speedup concerning the number of iterations. Since the number of function and derivative evaluations depends strongly on the number of iterations, and since for large problems the running time is highly correlated with the number of function and derivative evaluations this theorem clearly indicates that no substantial superlinear speedup is possible, assuming that the best-first strategy is used.

7. Summary

We presented a new approach to parallel verified global optimization. This approach uses a centralized mediator, responsible for the dynamic load balancing. For the first time to our knowledge a subdivision strategy that uses the gradient was applied in a parallel method. This subdivision strategy often provides a much faster algorithm in the serial as well as in the parallel case.

Measurements on a Connection Machine CM5 for a variety of test problems and the comparison with other parallel methods for verified global optimization showed the efficiency of the new parallel approach. For larger problems at least linear speedup was reached quite often. This also holds for two problems with infinitely many global minimum points.

Moreover, some theoretical results were given. It turned out that the same solution is found by the parallel method as by the serial one. It was shown that a good parallelization may be expected if the number of global minimum points is at least equal to p, the number of processors, and if these are well distributed over the starting box. A very important result of this paper is a theorem which clearly indicates that, applying the best-first strategy, no real superlinear speedup may be expected concerning the number of iterations. In most cases the running time

highly correlates with the number of iterations, so this will also be true for the speedup concerning the running time.

References

Alefeld, G., Herzberger, J. (1983), *Introduction to Interval Computations*, Academic Press, New York.

Berner, S. (1995), Ein paralleles Verfahren zur verifizierten globalen Optimierung, PhD Thesis, Bergische Universität GH Wuppertal, Germany.

Berner, S. (1996a), Parallel validated global optimization. To appear in *ZAMM*.

Berner, S. (1996b), A parallel method for verified global optimization, in Alefeld, G., Frommer, A., Lang, B.(eds.), *Scientific Computing and Validated Numerics*, Akademie-Verlag, Berlin, 200–206.

Berner, S. (1996c), New results on verified global optimization. Accepted for publication in *Computing*.

Csendes, T., Ratz, D., Subdivision direction selection in interval methods for global optimization. To appear in *SIAM Journal of Numerical Analysis*.

Eriksson, J. (1991), Parallel global optimization using interval analysis, Licentiate Thesis, University of Umeå.

Hansen, E. (1992), *Global Optimization Using Interval Analysis*, Marcel Dekker, New York.

Henriksen, T., Madsen, K. (1992), Use of a depth-first strategy in parallel global optimization, Tech. Report 92-10, Institute for Numerical Analysis, Technical University of Denmark, Lyngby.

Leclerc, A. (1993), Parallel interval global optimization and its implementation in C++, *Interval Computations*, 3, 148–163.

Klatte, R., Kulisch, U., Neaga, M., Ratz, D., and Ullrich, C. (1991), *Pascal-XSC — Sprachbeschreibung mit Beispielen*, Springer-Verlag, Berlin, Heidelberg.

Klatte, R., Kulisch, U., Lawo, C., Rauch, M., and Wiethoff, A. (1993), *C-XSC - A C++ Class Library for Extended Scientific Computing*, Springer-Verlag, Berlin, Heidelberg.

Moore, R. E., Hansen, E., Leclerc, A. (1992), Rigorous methods for global optimization, in Floudas, C. A., Pardalos, P. M. (eds.), *Recent Advances in Global Optimization*, Princeton University Press.

Neumaier, A. (1990), *Interval methods for systems of equations*, Cambridge University Press, Cambridge.

Ratschek, H., Rokne, J. (1988), New Computer Methods for Global Optimization, Ellis Horwood, Chichester.

Ratz, D., Csendes, T. (1995), On the selection of subdivision directions in interval branch-and-bound methods for global optimization, *Journal of Global Optimization*, 7, 183–207.

Rinnoy Kan, A. H. G., Timmer, G. T. (1987), Stochastic global optimization methods, Part II: Multi level methods, *Mathematical Programming*, 39, 57–78.

Smith, S. L., Schnabel, R. B. (1992), Dynamic scheduling strategies for an adaptive, asynchronous parallel global optimization algorithm, Tech. Report CU-CS-625-92, University of Colorado.

Törn, A., Žilinskas, A. (1989), *Global Optimization*, Springer-Verlag, Berlin, Heidelberg.

Walster, G., Hansen, E., Sengupta, S. (1985), *Test results for a global optimization algorithm*, in Boggs, P., Byrd, R., Schnabel, R. (eds.), *Numerical Optimization 1984*, SIAM, Philadelphia, 272–287.

Appendix

A. Some Test Problems

In the following some of the test functions used are described which are not very common.

MHL(parameter estimation problem of Moore, Hansen and Leclerc (Moore *et al.*, 1992))
Define data points (x_i, y_i), $i = 1, \ldots, n$, with $n = 81$ by

$$x_i = 4.0 + 0.1(i+1),$$
$$y_i = a_1 e^{-\left[\frac{x_i - u_1}{s_1}\right]} + a_2 e^{-\left[\frac{x_i - u_2}{s_2}\right]}$$

with

$$a_1 = 130.89 \qquad a_2 = 52.6$$
$$u_1 = 6.73 \qquad u_2 = 9.342$$
$$s_1 = 1.2 \qquad s_2 = 0.97$$

Minimize

$$f(a_1, a_2, u_1, u_2, s_1, s_2) = \sum_{i=1}^{n} \left(a_1 e^{-\left[\frac{x_i - u_1}{s_1}\right]} + a_2 e^{-\left[\frac{x_i - u_2}{s_2}\right]} - y_i \right)^2 .$$

Starting box: $[130, 135] \times [50, 55] \times [6, 8] \times [8, 10] \times [1, 2] \times [0.5, 1]$.

CSEN(parameter estimation problem of Csendes, cf. Csendes and Ratz)

$$f(x) = \sum_{i=1}^{6} \left[\left(yre_i - (x_1 + \frac{x_2}{\omega_i^{x_3}}) \right)^2 + \left(yim_i - (\omega_i \cdot x_4 - \frac{x_5}{\omega_i^{x_3}}) \right)^2 \right],$$
$$x \in \mathbb{R}^5,$$

with $yre = (5, 3, 2, 1.5, 1.2, 1.1)^T$,
$yim = (-5, -2, -1, -0.5, -0.2, -0.1)^T$,
$\omega = (0.05\pi, 0.1\pi, 0.15\pi, 0.2\pi, 0.25\pi, 0.3\pi)^T$.

Starting box: $[0, 1] \times [0, 1] \times [1.1, 1.2] \times [0, 1] \times [0, 1]$.

GEO1(problem from geodesy[1])

$$f(x) = (\sqrt{x_2^2 + x_3^2 - 2c_1 x_2 x_3} - s_1)^2 + (\sqrt{x_3^2 + x_1^2 - 2c_2 x_3 x_1} - s_2)^2$$
$$+ (\sqrt{x_1^2 + x_2^2 - 2c_3 x_1 x_2} - s_3)^2, \quad x \in \mathbb{R}^3$$

with

i	c_i	s_i
1	0.846735205	1871.10
2	0.928981803	1592.40
3	0.912299033	1471.90

Starting box: $[0, 3600] \times [0, 3520] \times [0, 3520]$.

GEO2(problem from geodesy)
$f(x)$ as in GEO1, $x \in \mathbb{R}^3$, with

i	c_i	s_i
1	0.740824038	6.2
2	0.817119474	5.0
3	0.737253644	6.3

Starting box: $[0, 8.68] \times [0, 9.24] \times [0, 8.68]$.

GEO3(problem from geodesy)
$f(x)$ as in GEO1, $x \in \mathbb{R}^3$, with $c_i = 0.766044443$, $s_i = 5.0$ for $i = 1, \ldots, 3$.

Starting box: $[0, 8.0]^3$.

HM1(problem of Henriksen and Madsen (Henriksen and Madsen, 1992))

$$f(x) = y^T A y - x_1 \text{ with } y_i = \sin(x_i), \quad A = \begin{bmatrix} 1 & -1 & & & 0 \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & -1 & 2 & -1 \\ 0 & & & -1 & 2 \end{bmatrix}, \; x \in \mathbb{R}^5$$

Starting box: $[-1, 1]^5$.

HM2(problem of Henriksen and Madsen (Henriksen and Madsen, 1992))
$f(x)$ as for problem HM1, $x \in \mathbb{R}^7$
Starting box: $[-1, 1]^7$.

HM3(problem of Henriksen and Madsen (Henriksen and Madsen, 1992))

$$f(x) = -\sum_{i=1}^{2} \sum_{j=1}^{5} j \sin((j+1)x_i + j), \; x \in \mathbb{R}^2$$

Starting box: $[-10, 10]^2$.

HM4(problem of Henriksen and Madsen (Henriksen and Madsen, 1992))
$f(x) = -\sum_{i=1}^{3}\sum_{j=1}^{5} j \sin((j+1)x_i + j),\ x \in \mathbb{R}^3$
Starting box: $[-5, 5]^3$.

HM5(problem of Henriksen and Madsen (Henriksen and Madsen, 1992))
$f(x) = \sum_{i=1}^{10}((e^{-t_i x_1} - e^{-t_i x_2}) - x_3(e^{-t_i} - e^{-10t_i}))^2, \quad t_i = \frac{1}{10}, \quad x \in \mathbb{R}^3$
Starting box: $[0.9,\ 1.2] \times [9,\ 11.2] \times [0.9,\ 1.2]$.

HM6(problem of Henriksen and Madsen (Henriksen and Madsen, 1992))
$f(x) = \sum_{i=1}^{10}((\ln(x_i - 2))^2 + (\ln(10 - x_i))^2) - (\prod_{i=1}^{10} x_i)^{0.2},\ x \in \mathbb{R}^{10}$
Starting box: $[3, 4]^5 \times [3, 6]^5$.

UEND1
$f(x) = (x_1 - x_2)^2,\ x \in \mathbb{R}^n$
Starting box: $[-2.0,\ 2.5]^2$.

UEND2
$f(x)$ as for problem SC2.7.
Starting box: $[0.\,11] \times [0,\ 11] \times [-3,\ 3]$.

Notes

[1] Problems GEO1, GEO2 and GEO3 are due to Prof. Dr. G. Heindl, Universität Wuppertal.