Interval Computations
No 3, 1994

Finding Local Extremal Points by Using Parallel Interval Methods

Erlang Lyager

Finding all extremal points of a n-dimensional function is usually quite difficult. By a special partitioning search, interval methods are used to ensure that no extremal points are overlooked. The extremal points are furthermore located precisely and unambiguously.

Although no predictable structure exists within the search, a parallel version shows almost linear speedup on a MIMD type architecture. The idea behind the parallel method seems generally applicable to problems having a large number of subtasks which show no discernible structure with respect to their order of appearance or amount of work.

Нахождение точек локальных экстремумов с использованием параллельных интервальных методов

Э. Люагер

Нахождение всех точек экстремума n-мерной функции представляет собой достаточно сложную задачу. Применение специального поиска с разбиением в интервальных методах может дать уверенность, что ни одна точка экстремума не была пропущена. Кроме того, местоположение точек экстремума устанавливается при этом точно и однозначно.

Хотя такой поиск не обладает предсказуемой структурой, его параллельная версия обеспечивает почти линейное ускорение быстродействия на архитектуре MIMD. Параллельные методы, по-видимому, хорошо применимы к любым задачам, состоящим из большого числа подзадач с непредсказуемым порядком появления или объемом работы.

1 Introduction

The use of interval methods for computing the range of a real-valued function is well known ([4] and [5]). When regarded as an optimization technique this corresponds to finding the global extremal values of the function, and usually the extremal points are found simultaneously.

In the opinion of the author, considerable interest is associated with the similar problem of finding all *local* extremal points (and extremal values). As an example, the existence and knowledge of local extremal points are important with respect to the physics of phase transitions. Given this viewpoint, an algorithm has been found that discovers all local and global extremal points. The algorithm is in essence a modified version of a similar algorithm used for finding the range of a function. The algorithm requires substantial computing time because of the use of interval methods, thus making the effort of constructing a parallel version worthwhile.

The method used to generate a parallel version of the algorithm is the second major theme of this paper (the first theme being the search for all extremal points). This method may possibly be of general interest, as it may be applied to a whole class of algorithms that are described by the following properties.

- The work is progressively subdivided into smaller (sub-)tasks, created during the operation of the algorithm.
- It is neither possible to predict the order of appearance of the tasks, nor the amount of work associated with a task.

1.1 Th

The algori
twice conti

The algorit
when f is r
a set of the

In the f
contained i

When we fi
able to find
is a continu
endpoints c
the global r

Our algo
sion F with
extension t
positive con

Here $\overline{f}(\underline{X})$
(i.e., max(b_i

In additi
extensions t
ist $\partial_\alpha F$ (th
of f) and ∂
derivatives c

In this p
interval exte
second deriv
interval ope

1.1 The problem

The algorithm that searches for local extremal points requires a C^2 (i.e. twice continuously differentiable) real-valued function f:

$$f : D \to \mathbf{R} \qquad D \subseteq \mathbf{R}^n,\ n > 0. \tag{1}$$

The algorithm then computes all extremal points to a given precision ε, when f is restricted to an arbitrarily closed n-dimensional interval $\underline{X}$ (i.e., a set of the type $[a_1, b_1] \times \cdots \times [a_n, b_n]$) which is a subset of the domain D.

In the following text, $I(D)$ will denote the set of all closed intervals contained in D. This denotation can be reformulated as follows:

$$\underline{X} \subseteq D,\ \underline{X} \text{ closed interval} \Leftrightarrow \underline{X} \in I(D). \tag{2}$$

When we find all extremal points (including the global ones), we are thus able to find the range $\overline{f}(\underline{X})$ $\left(= \{f(\vec{x}) \,|\, \vec{x} \in \underline{X}\}\right)$ of f on $\underline{X}$ (indeed, since f is a continuous function on a compact set $\underline{X}$, its range is an interval, and endpoints of this interval are the values of f at the global minimum and the global maximum points).

Our algorithm requires that f has an *inclusion isotone interval extension* F with the property that the smaller the intervals, the closer is this extension to the range. To be more precise, we require the existence of positive constants K and ν such that:

$$\forall \underline{X} \in I(D) : d\left(\overline{f}(\underline{X}), F(\underline{X})\right) \leq K w(\underline{X})^{\nu}. \tag{3}$$

Here $\overline{f}(\underline{X})$ is the *range* of f on $\underline{X}$, $w(\underline{X})$ is the width of the interval $\underline{X}$ (i.e., $\max(b_i - a_i)$), and d is defined as $d([a, b], [c, d]) = \max(|a - c|, |b - d|)$.

In addition, we require that first and second derivatives of f also have extensions that satisfy the same inequality: so, we assume that there exist $\partial_\alpha F$ (the *inclusion isotone interval extensions* of the first derivatives of f) and $\partial^2_{\alpha,\beta} F$ (the *inclusion isotone interval extensions* of the second derivatives of f).

In this paper, as *interval extensions* F, $\partial_\alpha F$ and $\partial^2_{\alpha,\beta} F$, we take *natural interval extensions* of the corresponding functions (f, the first, and the second derivatives of f). A natural extension is what we get if we use interval operations instead of all the elementary operations of f. Natural extensions give linear approximations to the range (in the sense that $\nu = 1$) if f is either constructed solely of rational expressions in the independent variables, or by also using additional functions with known *range functions.*

The very existence of a natural extension presupposes that we have an expression for f. So, this algorithm will not be applicable to the case when f is given implicitly (e.g., by an equation $F(\vec{x}, f) = 0$), and the algorithm for computing $f(\vec{x})$ for a given $\vec{x}$ is still to be found.

1.2 The same idea can be applied to a more general situation

In the previous subsection, we have described the problem that we will solve in this paper. It is worth mentioning, however, that a similar algorithm can be used to solve a more general problem:

- Since f is continuous, the extremal points and the range of f can be found not only for the domain D itself, but also for an arbitrary connected subset of D (as soon as this subset can be approximated by unions of finitely many closed intervals). This additional ability can be added at the expense of vastly increased computing time.
- We will describe our algorithm for the case when $\nu = 1$ and natural extensions are used. However, a similar idea works for arbitrary extensions for which the above inequality is true. Therefore, in principle, we can apply a similar algorithm to implicit functions (for which such an extension is often known).

1.3 What parallelism we will use

The diversity of methods used to implement parallel versions of the same algorithm on different parallel architectures, makes it difficult to consider several parallel architectures in one paper. So, in this paper, we will consider only one type of architecture: a MIMD (Multiple Instruction, Multiple Data) hypercube. Specifically, the algorithm is used on a *Intel hypercube.* The highlights of this parallel machine are:

- $N = 2^d$ complex processors, with the *dimension* $d = 0, 1, 2, 3, 4,$ or 5 at DIKU. Processors are of type Intel 80286 with a 80287 floating point coprocessor, and each processor runs a simple multitasking operating system.
- All processors have separate memory (approximately half a megabyte).
- Any processor is connected to d neighbors.
- Total number of processor interconnections is $\frac{1}{2}dN$, and the *diameter* of the hypercube (i.e., the smallest number of communication steps that enable to send a message from each processor to any other processor) equals d.
- Any two processors may exchange messages, but communication between neighbors is faster.
- A special *manager* processor is used to start the parallel programs; it is the only processor with I/O facilities. Any processor may communicate with the manager.

We make our algorithm efficient by utilizing the possibility of fast communication between neighbors.

2 How to find all extremal points using interval methods

The presentation of the algorithm for finding all extremal points, starts off by describing a method used to find the range of the function f (i.e. the global extremal points and values).

2.1 Computing the range of f

This computation is based on techniques presented in [3] and [5]. It consists of the following rules applied repeatedly, until no more intervals are left. The description focuses on the search for the global minimum points and value. The search for the global maximum points and value is similar, and it can be done at the same time as the search for the minimum.

1. The interval $\underline{X}$ is subdivided into smaller intervals $\underline{X}_i$ which in turn are subdivided further. For any subdivision $\underline{X}_i$, $i = 1, 2, \ldots, l$ of $\underline{X}$, one gets by the use of (3):

   $$d\left(\overline{f}(\underline{X}), \bigcup_{i=1}^{l} F(\underline{X}_i)\right) \le K \max_{i=1}^{l} w(\underline{X}_i)^{\nu} \tag{4}$$

   which shows that $\overline{f}(\underline{X})$ may be approximated to within an arbitrary precision, by using a sufficiently detailed subdivision.

   The subdivision is constructed by "halving" the intervals, thereby (since $\nu = 1$) also halving the distance between $\overline{f}(\underline{X})$ and the approximation of the range for each subdivision. The subdivision method is therefore termed *bisection.*

2. It is not necessary to subdivide all intervals every time because a more efficient way may be achieved [5], just by subdividing the intervals $\underline{X}_p$ for which $F(\underline{X}_p)$ contains the minimum value.

   This is in practice done by subdividing the interval $\underline{X}_p$ which has the smallest lower bound of the minimum value, i.e.:

   $$\forall 1 \le i \le l : F(\underline{X}_p)_{inf} \le F(\underline{X}_i)_{inf} \tag{5}$$

   (here Y_{inf} denotes the lower endpoint of an interval Y). The interval $\underline{X}_p$ is replaced by its subintervals. This procedure leads to a detailed subdivision near the extremal points and to a coarse subdivision far away from the extremal points. Because the widths of the intervals replacing $\underline{X}_p$ are smaller than $w(\underline{X}_p)$, and because on every stage of an algorithm, there are only finite intervals involved, this special method of subdividing intervals eventually produces a better approximation for $\overline{f}(\underline{X})$.

3. Some subintervals can be discarded because they can not contain a global minimum point. This is achieved by maintaining an *upper bound for the minimum value.* To get such an estimate, we can, e.g., for each interval $\underline{X}_i$, evaluate the value of f at an arbitrarily chosen point from $\underline{X}_i$ (for example, at the midpoint of $\underline{X}_i$); the smallest

of these values can serve as the desired *upper bound.* If the lower endpoint $F(\underline{X}_p)_{inf}$ $F(\underline{X}_p)$ of an interval $F(\underline{X}_p)_{inf}$ is greater than the currently best known *upper bound for the minimum value*, then this interval can be discarded. The reason for this is that this interval can not contribute to the determination of the global minimum value, which is known to be smaller than any value in the interval.

The *upper bound for the minimum value* is thus calculated by:

$$F_{inf,sup} := \min_{i=1}^{l} f\big(m(\underline{X}_i)\big) \tag{6}$$

where $m(\underline{X}_i)$ denotes the midpoint of $\underline{X}_i$. The condition under which an interval $\underline{X}_p$ is discarded can be rewritten as follows:

$$F_{inf,sup} < F(\underline{X}_p)_{inf} \Rightarrow \text{ discard } \underline{X}_p. \tag{7}$$

4. Some other subintervals can be discarded by use of the monotonicity of f. For example, if for an interval $\underline{X}_i$, f is strictly increasing in α, then the global minimum can occur only at the points $\in \underline{X}_i$ for which α takes the smallest possible value. In other words, if $\underline{X}_i = \cdots \times [\alpha^-, \alpha^+] \times \cdots$, then the global minimum on $\underline{X}_i$ can be attained only for $\alpha = \alpha^-$. Such points are on the edge of $\underline{X}_i$. Therefore, they either belong to some other subinterval as well (so this interval $\underline{X}_i$ can be safely discarded), or this edge coincide with the edge of the initial interval $\underline{X}$.

   So the interval $\underline{X}_i = X_i^1 \times \cdots \times X_i^n$ is discarded or reduced if:

$$\begin{aligned} &\partial_\alpha F(\underline{X}_i) > 0 \Rightarrow \\ &\quad (X_{i,inf}^\alpha \neq X_{inf}^\alpha \Rightarrow \text{discard } \underline{X}_i) \vee \\ &\quad (X_{i,inf}^\alpha = X_{inf}^\alpha \Rightarrow \text{reduce } \underline{X}_i \text{ to } \underline{X}_i^{\alpha-left}) \end{aligned} \tag{8}$$

   where $\underline{X}_i^{\alpha-left} = X_i^1 \times \cdots \times [X_{i,inf}^\alpha, X_{i,inf}^\alpha] \times \cdots \times X_i^n$ (and a similar rule if $\partial_\alpha F(\underline{X}_i) < 0$).

5. The convexity of f may also be used to discard intervals because if f is convex (i.e. $\partial^2_{\alpha,\alpha} F(\underline{X}_i) < 0$) in some coordinate α, the minimum can only be found on the edge of $\underline{X}_i$. Just like in the case of monotonicity,

this can only happen if the edge of $\underline{X}_i$ in coordinate α is a part of the edge of $\underline{X}$ in coordinate α. Thus:

$$\begin{aligned}&\partial^2_{\alpha,\alpha}F(\underline{X}_i) < 0 \Rightarrow \\ &\quad \big((X^\alpha_{i,inf} \neq X^\alpha_{inf}) \wedge (X^\alpha_{i,sup} \neq X^\alpha_{sup}) \Rightarrow \text{discard } \underline{X}_i\big) \vee \\ &\quad \big(X^\alpha_{i,inf} = X^\alpha_{inf} \Rightarrow \text{reduce } \underline{X}_i \text{ to } \underline{X}_i^{\alpha-left}\big) \vee \\ &\quad \big(X^\alpha_{i,sup} = X^\alpha_{sup} \Rightarrow \text{reduce } \underline{X}_i \text{ to } \underline{X}_i^{\alpha-right}\big).\end{aligned} \tag{9}$$

If, however, $\partial^2_{\alpha,\alpha}F(\underline{X}_i) \geq 0$, nothing can be done.

6. The final method is the Krawczyk-method [2], a modified Newton-Raphson search for the zero point of $\partial_\alpha f$. The method applies the Krawcyzk operator:

$$\underline{K}(\underline{x}, \underline{X}) = \underline{x} - \underline{\underline{Y}}\partial F(\underline{x}) + \big(\underline{\underline{E}} - \underline{\underline{DR}}(\underline{X})\big)(\underline{X} - \underline{x}) \tag{10}$$

where $\underline{\underline{Y}}$ is any regular matrix, $\partial F(\underline{x})$ is the vector of $\partial_\alpha F(\underline{x})$, $\underline{\underline{DR}}(\underline{X})$ is the Jacobi interval matrix of $\partial^2_{\alpha,\beta}F(\underline{X})$, $\underline{\underline{E}}$ is the unity matrix, and $\underline{x}$ is any point in the interval $\underline{X}$. For the fastest convergence, one chooses:

$$\begin{aligned}\underline{x} &= m(\underline{X}) \\ \underline{\underline{Y}} &= m\big(\underline{\underline{DR}}(\underline{X})\big)^{-1}.\end{aligned} \tag{11}$$

The Krawczyk iteration is performed by:

$$\underline{X}_i' = \underline{X}_i \cap \underline{K}\big(m(\underline{X}_i), \underline{X}_i\big). \tag{12}$$

The Krawczyk operator has the following additional important properties: if $\underline{K}(\underline{x}, \underline{X}) \subseteq \underline{X}$, then precisely one extremal point exists in $\underline{X}$; and if $\underline{K}(\underline{x}, \underline{X}) \cap \underline{X} = \emptyset$, then $\underline{X}$ contains no extremal points. This last property provides us with an additional rule for discarding intervals.

When $\underline{X}_i$ is discarded by the use of this new rule, care must be taken to keep edges of $\underline{X}_i$ that coincides with the edges of $\underline{X}$, just like when using the monotonicity or convexity of f to discard intervals.

7. An interval $\underline{X}_i$ is declared to be a result, if:

$$\begin{aligned} &\big(w(\underline{X}_i) < \varepsilon \wedge \text{ the last application of the Krawczyk} \\ &\text{method did not reduce } \underline{X}_i\big) \\ &\qquad\qquad \Rightarrow \underline{X}_i \text{ is a result interval} \end{aligned} \tag{13}$$

for given precision ε. This combination criterion may seem strange, but experience shows that the ordinarily used criterion: $w(\underline{X}_i) < \varepsilon$ too often leads to inaccurate determination of the extremal points.

When no more intervals are left, the results are some intervals $\underline{X}_i^{res}$ containing the identified candidates for extremal points. These intervals give bounds for the global minimum value:

$$\begin{aligned} &\min_i F(\underline{X}_i^{res})_{inf} \text{ is the lower bound} \\ &F_{inf,sup}(\text{from Step 3}) \text{ is the upper bound.} \end{aligned} \tag{14}$$

2.2 Finding all extremal points of f

The search for all extremal points is difficult because any subinterval may conceal an extremal point. The key idea for the algorithm to overcome this difficulty, comes from the observation that Steps 2 and 3 above play important roles with respect to the decrease of the amount of work done, for finding the global extremal points and values. Unfortunately these two methods are specifically concerned with the global extremal value. This necessitates that the two methods should be *localized*, i.e. changed to be able to deal with local extremal points and values.

This is achieved by using the convexity/concavity of f to determine *areas* within $\underline{X}$ which contain *at most* one extremal point.

$$\begin{aligned} &\forall 1 \le \alpha \le n : \partial^2_{\alpha,\alpha} F(\underline{X}_i) < 0 \\ &\qquad \Rightarrow \underline{X}_i \text{ contains at most one maximum;} \\ &\forall 1 \le \alpha \le n : \partial^2_{\alpha,\alpha} F(\underline{X}_i) > 0 \\ &\qquad \Rightarrow \underline{X}_i \text{ contains at most one minimum;} \\ &\exists 1 \le \alpha \le n, 1 \le \beta \le n : \partial^2_{\alpha,\alpha} F(\underline{X}_i) < 0 \wedge \partial^2_{\beta,\beta} F(\underline{X}_i) > 0 \\ &\qquad \Rightarrow \underline{X}_i \text{ contains no extremal points.} \end{aligned} \tag{15}$$

Note. Ideally the convexity/concavity/saddle point of f should be determined by calculating the interval eigenvalues of the Jacobi interval matrix, but this requires much more time than this simpler approach, and the results are practically the same (only for a few functions eigenvalues lead to drastically better results).

Thus each subinterval belongs to a certain kind of *area* which carries a new characterization, i.e. what kind of extremal point it may contain. There are five different types.

1. Unknown—initially, all subintervals belong to this class.
2. Local Maximum—when a subinterval is determined to contain at most one maximum point (by the first rule above), it constitutes a new *area*. Within this new area the algorithm only searches for a local maximum (or global extremal points).
3. Local Minimum—as for "Local Maximum".
4. Global Maximum—when a subinterval is reduced, it may leave behind edges in common with the initial interval $\underline{X}$ (as in Step 4). Within these edges of $\underline{X}$, the algorithm only searches for a global maximum point.
5. Global Minimum—as for "Global Maximum".

When a subinterval is characterized to be of the types 2 to 5, it spawns a new *area* of the same type. All subintervals from this area are of the same type therefore, it is no longer necessary to compute the characterization of these subintervals.

It is this characterization based upon interval techniques which ensures that the search doesn't miss any interval containing extremal points. However, the characterization may fail (as discussed in the note above, and demonstrated in Section 4), causing too many candidate extremal points to be found.

Within an area, it is possible to use the algorithm for the computation of the range of f because here the local extremal point is also a global extremal point (disregarding the edges). The *range algorithm* of Section 2.1 can be used with the following minor changes.

1. The
2. The the
3. Disc *valu* intro

and

4. The
5. Con kno
6. The
7. The char

3 Th

The natu done for expressio time spen of subinte

There subinterv for the di knowledg ready me tremal po tribute th jobs are g

1. The subdivision is the same way as before (no changes in that).
2. The choice of the interval to subdivide is made as previously, with the only difference that we are choosing an interval within each *area*.
3. Discarding the intervals by use of the *upper bound for the minimum value* (or the lower bound for the maximum value) is modified by introducing *area* specific upper and lower bounds:
$$F^{area}_{inf,sup} \quad \text{and} \quad F^{area}_{sup,inf} \tag{16}$$
and using these with method 2.
4. The use of the monotonicity of f is similar (unchanged).
5. Convexity/concavity is not used when the type of an interval is known.
6. The same Krawczyk method is applied.
7. The criterion for determining a result interval is the same (unchanged).

3 The parallel search

The natural job unit in the search for extremal points is the calculations done for a single subinterval. This is true, if calculation of the interval expressions for F, $\partial_\alpha F$, and $\partial^2_{\alpha,\beta}F$ are time consuming, compared to the time spend doing the "housekeeping" of the algorithm, and if the number of subintervals is large.

Therefore, the responsibility of the parallel version is to distribute the subintervals to the processors in the best possible way. However, no plan for the distribution can possibly be made in advance because this requires knowledge of the position of the extremal points (since, as we have already mentioned, most subintervals will be generated around these extremal points). One is thus left with only parallel algorithms which distribute the computing job during the operation of the algorithm (when the jobs are generated).

During the project which formed the basis for this paper, we examined several different parallel versions of the above-described algorithm. One of them turned out to work well (i.e., to find all the extremal points) in most cases. In some cases, other versions were better. However, we will present arguments in favor of this chosen version that strengthen our belief that this is the version to be recommended in the general case. the belief that it is a generally applicable and efficient parallel version.

3.1 The details of the parallel version

All parallel versions that we have checked follow the algorithm described in the previous section. The difference is solely in the distribution of the subtasks (i.e. the subintervals) between the hypercube's processors. In the version that turned out to be the best, the distribution is performed by using the following rules.

1. If any processor is idle, it requests a subtask from each of its d neighbors.
2. If any processor receives a request for a subtask, it determines whether or not it has more than one subtask. If this is the case, it sends one of its subtasks to the requesting processor (what exactly subtask it will send is described by Rule 3). If the processor does not have more than one subtask, it registers the request for later treatment.
3. In case that the requested processor has several subtasks, it chooses one by the following order of priority.
   (a) Subintervals of unknown type have first priority.
   (b) Subintervals of known type, but not belonging to the same *area* as the currently processing subinterval of the processor, have second priority.
   (c) Other subintervals are of third priority.
4. When a subtask is transferred, is always carries the following information.
   (a) The subinterval itself.

   (b) The area of the subinterval, including the best known information about the area. This includes the *lower bound of the maximum value* and the *upper bound of the minimum value* for the area.

   (c) The best known *global information*, i.e., the *lower bound of the global maximum value* and the *upper bound of the global minimum value.*

5. On reception of a subtask, the information on the *area* and the global information is used to update the corresponding information in the receiving processor.

6. Upon discovery of a new *area*, this is identified by:

   (Processor-number, Sequential-number)

   where the *Processor-number* is an identification of the processor (from 0 to $2^d - 1$), and the *Sequential-number* is a count of the number of *areas* this processor has discovered. This identification follows the area and its subintervals when the subintervals are transferred to other processors.

7. When a result interval is found, its corresponding subtask is sent to the *manager* processor which applies the method of discarding the subintervals by use of the upper/lower bound for minimum/maximum value (Step 3 of Section 2.1).

8. The parallel algorithm terminates when no processor has any subintervals left, and when no messages are in transit between two processors. This situation is determined using an algorithm described in [1].

3.2 Comments on the rules of the parallel version

These rules have the following important characteristics that contribute to the validity and/or to the efficiency of this parallel version.

1. This rule more or less guarantees that the scaling behavior of the parallel version with total processor number N, does not make the

parallel program unusable if it is used on a hypercube with a large number of processors. This can be demonstrated by assuming the following.

(a) The total number of subintervals in the problem, is assumed to be proportional to the number of processors N because if the amount of work is fixed with changing N, the performance of most parallel algorithms deteriorates with increasing N. This assumption gives a constant amount of work per processor, whatever value of N.

(b) The total number of messages (the sum of the number of requests and the number of transferred subtasks) is assumed to be proportional to the total number of subintervals, i.e. proportional to N.

Given these assumptions, the number of requests per processor is not dependent upon N. Thus the communication facilities of the hypercube are not saturated with increasing N.

2. The requesting processor is given a subtask even when a subtask is not available initially (i.e., at the moment of time when the request was received). A very long time may pass before a subtask is sent to the requesting processor, and meanwhile, this requesting processor may have received several subtasks from some of its other neighbors.

   This rule keeps the subtasks *flowing* among the processors, and this *flow* is a desired feature (see the comment for Step 4).

3. The prioritized selection of the subtask to be transferred to another processor, is based upon the wish that the work done for a single *area* should not be spread over more than a few processors (if possible).

   The justification for this involves the determination of the *lower/upper* bounds of the *maximum/minimum* values. Ideally these bounds should be globally available to all processors, but global values are not easily obtained in a MIMD architecture. The second best solution keeps the potentially global information on as few processors as possible, especially in combination with Step 4. Notice, that this only solves the problem of global information for the *area* specific values of the *lower/upper* bounds. The global *lower/upper* bounds will not be well known throughout the processors.

4. The rule of always transferring the currently best known values of the *area* specific and the global *lower/upper* bounds are based on the same wish as for Step 3. It is done in order to (hopefully) maintain an approximation of the idea that these values are global items.

5. This rule is just the updating part of Step 4.

6. The identification of the *area* is necessary, in order to be able to globally identify which area a certain subinterval belongs to. The construction of the identification is a kind of *timestamping*, ordinarily used with distributed systems.

7. Several processors may find the same extremal point, situated on the edge of $\underline{X}$. They may even find extremal points on the edge of $\underline{X}$ that are not global extremal points (because of the lack of knowledge of global *lower/upper* bounds, see the comment for Step 3). This requires that a central agent eliminates the duplicates and the spurious extremal points.

8. The termination detection is initiated (arbitrarily) by processor number zero.

   Notice that the termination detection algorithm from [1] requires instantaneous communication. This restriction may be alleviated, if the *termination token* of the algorithm also carries the information:

   - Total number of messages sent *minus* number of messages received, for all processors up to, and including, the current processor.

   If this number is zero when calculated in processor zero (and the other criteria of the termination detection algorithm are satisfied), the parallel program is terminated. This assumes that all messages are single messages.

4 Results

In cases where the Krawczyk method is able to reduce the subintervals, the determination of all extremal points succeeds, and the extremal points are found to within two decimal digits of the computer's floating point precision (such an accuracy is due to our choice of a special criterion for determining a result interval). Furthermore, no "spurious" extremal points are found (i.e. points that the algorithm claims are extremal points, while in reality they are not). If the Krawczyk method does not work (e.g. if an extremal point is in the vicinity of a point where the Jacobi matrix is singular), the extremal points are only determined with precision ε, which results in inaccurate and ambiguous determination.

In a few cases, like with the function:

$$f(x,y) = xy \qquad \underline{X} = [-1,1] \times [-1,1] \tag{17}$$

the algorithm never decides whether $(0,0)$ is an extremal point or not, because no area is found by the use of (15).

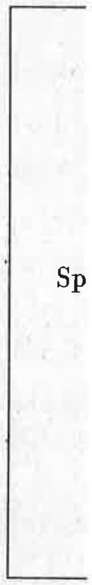
The following function:

$$\begin{aligned} f(x,y) &= e^{-1/x}\sin(-1/x) + e^{-1/y}\sin(-1/y) \\ \underline{X} &= [0.01,10] \times [0.01,10] \end{aligned} \tag{18}$$

serves as an example of the normal functioning of the algorithm. This function has 256 minimum points and 256 maximum points in $\underline{X}$. The distance between the (local) extremal points decreases from around 0.4 to approximately $3 \cdot 10^{-4}$ with decreasing x, y, and the corresponding extremal values decrease from ≈ 1 to approximately $4 \cdot 10^{-38}$.

In such a complicated situation, the algorithm determines all 512 extremal points, and no others, when used with precision $\varepsilon = 10^{-5}$. During the search for the extremal points, approximately 9500 subtasks are generated, each requiring approximately one second of CPU time. This number is sufficiently large to utilise all 32 processors fully. The following figure shows the parallel *speedup*, calculated by:

$$\textit{Parallel Speedup} = \frac{\textit{Computation time for one processor}}{\textit{Computation time for } n \textit{ processors}}. \tag{19}$$



The al
as long as
each dema

Furthe
subtasks t
cated indi
directly wl
of the algo

This fu
many large
more essen
faster. As

$f(x,y)$

This funct
mately 0.4
only 28. T
before. He
are more
subtasks u

subintervals,
remal points
oating point
criterion for
remal points
points, while
vork (e.g. if
bi matrix is
ion ε, which

(17)

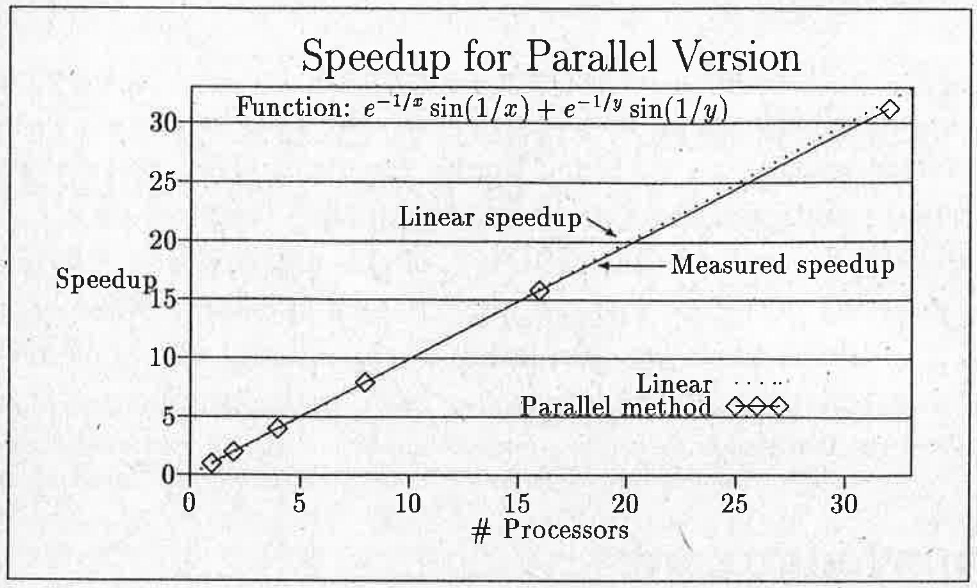
oint or not,

(18)

thm. This
in $\underline{X}$. The
ound 0.4 to
g extremal

all 512 ex-
$^{-5}$. During
are gener-
his number
ving figure

. (19)



The almost linear speedup is a general feature of the parallel version as long as the function in question generates a large number of subtasks, each demanding much CPU time.

Furthermore, the parallel version is quite successful in distributing the subtasks to the processors, without using too much time. This was indicated indirectly by an almost linear speedup, and it was also confirmed directly when we monitored the processors' activity during the execution of the algorithm.

This function f is well suited for our parallel algorithm, because it has many large subtasks. It is natural to expect that the speedup will deviate more essentially from the ideal linear one, if the function can be calculated faster. As an example of such function, we took the following function:

$$f(x,y) = \sin(-1/x) + \sin(-1/y) \qquad \underline{X} = [0.01, 10] \times [0.01, 10]. \tag{20}$$

This function generates 2000 tasks; each of these tasks requires approximately 0.4 CPU seconds. For 32 processors, the speedup in this case is only 28. The reason for this is as follows: computing $f(\vec{x})$ is faster than before. Hence, the number of subtasks per unit time increases. Since there are more subtasks per unit time, the mechanism for distributing these subtasks uses a larger part of the computation time.

5 Summary

This paper describes an interval based algorithm for finding all extremal points of an explicitly given real-valued n-dimensional function. The algorithm guarantees that no extremal points are overlooked, and determines the extremal points accurately and unambiguously (in most cases).

We propose a parallel implementation of this algorithm on a hypercube that (in general) performs with an almost ideal speedup. Namely, linear speedup is attained when the calculation of the interval values of the function is not too simple, and the function is sufficiently complicated (i.e., has many extremal points).

Acknowledgments

Thanks to Stig Skelboe (DIKU) for suggestions and corrections.

References

[1] Dijkstra, E. W., Feijen, W. H. J., and Gasteren, A. J. M. *Derivation of a termination detection algorithm for distributed computations.* Information Processing Letters **16** (1983), pp. 217–219.

[2] Krawczyk, R. *Newton-Algorithme zur Bestimmung von Nullstellen mit Fehlerschranken.* Computing **4** (1969), pp. 187–201.

[3] Moore, R. E. *On computing the range of a rational function of n variables over a bounded region.* Computing **16** (1976), pp. 1–15.

[4] Ratschek, H. and Rokne, J. *Computer methods for the range of functions.* Ellis Horwood Limited, 1984.

[5] Skelboe, S. *Computation of rational interval functions.* Bit **14** (1974), pp. 87–95.

Received: October 25, 1992
Revised version: October 17, 1993

Department of Computer Science
University of Copenhagen (DIKU)
Copenhagen, Denmark
E-mail: `erling@diku.dk`

The si
tions i
the pr
rounde
tests d
most l
numbe
proces

Рассм
линей
ралле
ем ин
жения
эффек
лено н
до нач