

# PMB-Parallel Multidimensional Bisection

by

**William Baritompä**

*Department of Mathematics and Statistics,  
University of Canterbury, Christchurch, New Zealand*

and

**Sami Viitanen**

*Department of Computer Science,  
Åbo Akademi University, Åbo, Finland*

No. 101

September, 1993

**Abstract** – A master-slave parallel framework of the multidimensional bisection global optimization method of Wood is presented. Various heuristics for minimizing inter-transputer communication are tested on the Hathi-2 system using an OCCAM-2 implementation. For the simple test function used, a modest speed up of 6 was attained with 32 transputers. For more computationally intensive functions, we expect higher efficiencies.

**Abstrakt** – En metod för parallellisering av Woods flerdimensionella bisektionsmetod för global optimering presenteras. Olika heuristiska prövas för att minimera inter-transputer kommunikation på Hathi-2 genom att utnyttja en OCCAM-2 implementation. För den enkla testfunktion som utnyttjas uppnås en blygsam speed-up på 6 då 32 transputers används. För funktioner som är tyngre att evaluera uppskattar vi högre effektivitet.

**Keywords**—Global optimization, parallel algorithms, multidimensional bisection.

## PMB-Parallel Multidimensional Bisection

**Abstract**—A master-slave parallel framework of the multidimensional bisection global optimization method of Wood is presented. Various heuristics for minimizing inter-transputer communication are tested on the Hathi-2 system using an OCCAM-2 implementation. For the simple test function used, a modest speed up of 6 was attained with 32 transputers. For more computationally intensive functions, we expect higher efficiencies.

**Abstract**—En metod för parallellisering av Woods flerdimensionella bisektionsmetod för global optimering presenteras. Olika heuristiska prövas för att minimera inter-transputer kommunikation på Hathi-2 genom att utnyttja en OCCAM-2 implementation. För den enkla testfunktion som utnyttjas uppnås en blygsam speed-up på 6 då 32 transputers används. För funktioner som är tyngre att evaluera uppskattar vi högre effektivitet.

**Key words:** Global optimization, parallel algorithms, multidimensional bisection.

### 1. INTRODUCTION

This paper discusses parallel implementation of multidimensional bisection (MB) global optimization methods of Wood [6]. These multidimensional bisection methods have the salient features of the usual “root finding” bisection, in that they produce a nested family of bracketing regions for the global minimum. They can be viewed as a geometric realization of Piyavskii’s general approach [4] which uses a lower envelope of a function to estimate the global minimum, although as pointed out in [1], they can be used without building lower envelopes. Further details of these techniques are given in [6, 7, 2]. The theory, context and performance of the optimization techniques can be found in [6, 7, 8, 1]. A fast sequential implementation ([2]) requires the use an appropriate multidimensional data structure. An alternate way of speeding up execution is by a parallel implementation.

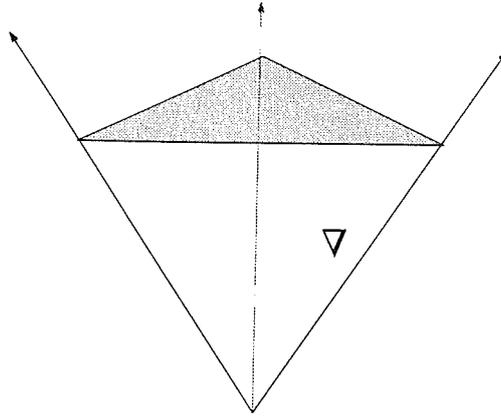
The approach we take here is to split the bracketing region between transputers. Each transputer then works on its own separate bracketing region, thus achieving parallel function evaluations and keeping track of smaller lists. Since the lists are shorter, the more complicated data structure required by the sequential version can be replaced by simple linear structures. Using a multi-transputer system, we implement PMB, a parallel version of MB and explore various heuristic strategies for communication and sharing of the work load.

Section 2 provides a brief description of sequential multidimensional bisection. Section 3 discusses a basic master/slave parallel implementation. Section 4 gives details of inter-transputer communication. Section 5 gives test results. Section 6 gives the basic conclusion and discusses future directions.

### 2. SEQUENTIAL MULTIDIMENSIONAL BISECTION

A brief review of multidimensional bisection (MB) extracted from [6] and [2] is given here. Refer to [6, 8, 2] for more details. The problem is the following: given  $f : R^n \rightarrow R$  and  $K$  a compact domain in  $R^n$ , find the points on the graph of  $f$  where  $\min f(x)$  over  $x \in K$  is realized. It is assumed that a constant  $M$  is known for which the function  $f$  belongs to  $L(M)$ , the set of Lipschitz continuous functions.

The approach taken by Wood starts by bracketing all global minima in a simplex. At each iteration regions are cut away from this initial simplex in such a way as to leave a *system* of simplexes, the union of which gives an improved bracket.

Figure 1. Simplex Based Cone  $\nabla$ 

Referring to Figure 1, let  $\nabla$  be a cone with a simplex base in  $R^{n+1}$  of slope  $M$  (i.e. contained in  $\{(x, y) | y \geq M \|x\|\}$ ). The two mathematical facts (see [6]) that insure a convergent algorithm are the following. For each  $(x, y)$  on the graph of  $f$ , (1) no point inside  $(x, y) - \nabla$  and (2) no point above  $(x, y)$  can be a global minimum of  $f$ .

These two facts allow an algorithm to be set up in a very simple way. A sequential version of multidimensional bisection is described now.

At the outset the initial system consists of one standard simplex,  $T_0$ , which brackets all global minima over  $K$ . Here a *standard simplex* is a translate of a cap of the cone  $\nabla$ . For each function evaluation, cut from every simplex in the system the interior of  $-\nabla$ , with apex moved to the evaluation point on the graph of  $f$ . Also cap all the simplexes at the height of the lowest known evaluation. These processes are termed *reduction* and *elimination* in [6], or *cutting* and *capping* in [1]. When these are done to a standard simplex, at most  $n + 1$  standard simplexes, of smaller height than the original are left. This key process for one standard simplex is shown in Figure 2.

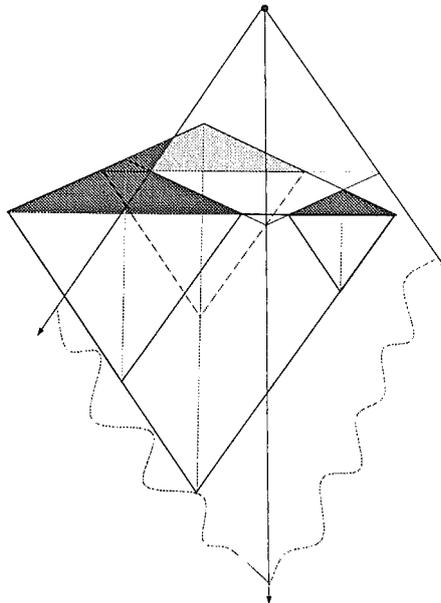


Figure 2. Cutting and Capping A Standard Simplex

After each iteration, the algorithm brackets all global minima over  $K$  in the union of the standard simplexes belonging to the current updated system. Figure 3 shows such a system for a function of two variables. All simplex tops, shaded in the figure, lie at the height of the least

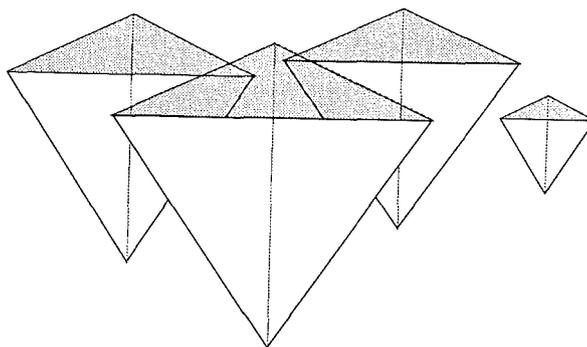


Figure 3. System of Standard Simplexes

evaluation to date. The process continues until the maximum height of all simplexes in the system (the *variation*) is small enough. Properties (1) and (2) above guarantee no global minimum are removed.

Figure 4 gives a geometric example of the process in the one dimensional case with  $M = 1$ . Cutting and capping remove the two regions at the evaluation and leave an improved bracket (shown outlined in bold). In this example the bracket begins as the union of a system of four simplexes (triangles outlined lightly). The geometric process is realized by removing the regions from each to get the updated system. Here the leftmost triangle is affected only by capping, the middle two produce four new smaller triangles, and the rightmost is completely removed.

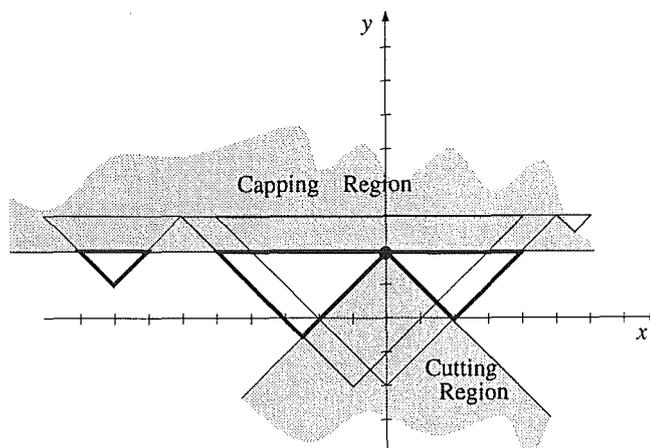


Figure 4. A One Dimensional Example

The example illustrates the procedures needed in order to implement the algorithm. The effect of cutting and capping must be described and ways to eliminate inefficient representation must be handled. The detailed formulae for these procedures can be found in [6, 8, 2].

Various strategies for choosing the next function evaluation lead to different versions of MB [6, pg. 165]. Choosing the projection of the apex of the simplex with the biggest height (i.e. using the deepest simplex) gives the multidimensional bisection analogue of the Piyavskii-Shubert algorithm [4, 5]. This is shown to be one-step optimal in [6], but some small simplexes in the bracketing system may never be affected by the cutting process, thus in the limit, the bracket may not be the set of global minimum. If it is important to find the location of the global minimum, using the oldest simplexes first prevents the cutting process from missing some simplexes. This gives the original MB algorithm presented in [7].

A dual approach ([2]) leads to a simplification of multidimensional bisection, and also implements generalizations (see [1]). Capping can be done with convex regions other than hyper-planes. Also the bracket can be built up from convex bodies other than standard simplexes. For this reason in the rest of this paper we refer to systems of *bodies* rather than simplexes.

In order to implement the algorithm efficiently, some refinement is necessary. It is desirable to describe a system of bodies with the minimal amount of storage requirements. There is no need to represent empty bodies or redundant bodies properly contained inside other ones<sup>1</sup>. A *minimal uniform system*, which contains only representations of non-empty bodies and has the property that no body is properly contained in any other provides a more parsimonious description. A simple test for these cases is described in [2].

The process of reduction need only be applied to those simplexes that actually meet the removal cone. Such simplexes in the system are the only ones necessary to look at when implementing the cutting process. The following terminology (inspired during a visit to the salmon ladder at Seattle's aquarium) is useful. Given an evaluation, we say a member of the system is a *spawner* if it meets the removal cone. The *spawn* of a spawner are all the non-empty simplexes produced by the cutting process.

#### *MB algorithm*

<i>Initialization</i>	Choose a body to make up initial system
<i>Get Next Point</i>	Find the (oldest) body with biggest variation.
<i>Capping</i>	Update the new system top.
<i>Tidy Up</i>	Remove any empty bodies.
<i>Cutting</i>	Find all the spawners and compute their spawn.
<i>Tidy Up</i>	Remove any empty or redundant bodies.
<i>Stopping Test</i>	If the variation is small enough, terminate.

For efficient sequential implementation, it is necessary to use an appropriate data structure (see [2]). This is particularly crucial for finding the spawners, as the “brute force” method of inspecting each body is costly for large systems. However if the system never gets large, sophisticated data structures are not needed as brute force is sufficient.

### 3. BASIC PARALLEL MB

As we have seen above, MB consist of iterating through capping-cutting-tidy up cycles until a global solution is found. A parallel version of MB (PMB) will share the system among many processors. We relax the condition that a system describes a bracket. As a result of one iteration, one or more of the following things may happen to the system being handled by a given processor:

- A new system top (best result) is found.
- One or more “large” bodies are replaced by one or more smaller bodies.
- Empty or redundant bodies are removed.

As a result the system of bodies for a given processor may either get bigger or smaller. It could even disappear altogether in the case the system being handled by the given processor does not bracket the global optimum. A parallel algorithm need handle two important possibilities: (1) broadcast new system top information when it occurs and (2) deal with processors that are running out of work.

The reason for the first possibility is that every processor works independently from the others, so the communication of a new best result prevents a processor from working on bodies that are now known to be “above” the system top and therefore should be removed from its system. Otherwise some processors will be doing wasted work. Therefore, whenever a processor finds a new (better) system top, this value has to be broadcast through the network of processors.

The second possibility is more complicated. Every so often a processor will run out of bodies. Due to the fact that the total system (all bodies on all processors) is not stored in any one place, it is impossible for this processor to know whether a global solution has been reached. It is more likely that other processors will still have a lot of work left. The fact that this processor has no

---

<sup>1</sup>Overlapping bodies which lead to redundancies, usually occur only when the dimension is greater than 1.

bodies left is just a result of either finding a new system top which results in the pruning of all bodies in this processor (and possibly several other processors), or the cutting process removes all bodies the given processor is working on because it happens to be working with areas that do not contain the global optimum. It would be inefficient to let a processor stay out of work. Instead this processor should ask for more work (new bodies) from its neighbours which presumably still have some work to give.

Therefore for PMB we set up a mechanism for communication capable of handling the following three cases:

- The broadcast of a new system top.
- A request for new bodies (one or more).
- The answer to a request which can be either negative (a tag signalling that the requestee has no bodies to give) or positive (a mini-system of bodies).

The details of this mechanism depends on the actual topology chosen to connect the processors to each other. The next section presents the details of a transputer implementation.

#### 4. A TRANSPUTER IMPLEMENTATION

Since a transputer can only be connected to four other it follows that a broadcast necessitates forwarding of messages. If all four connections are used to connect the transputers in a torus then the number of forwarding "middle-men" needed for a broadcast is only 3 for 16 transputers and 4 for 30 transputers (arranged in a 5x6 torus). This can be compared to 14 and 28 respectively for a single-linked ring and 7 and 14 respectively for a doubly-linked ring. Therefore a torus was chosen as the topology to connect the transputers. We introduce some notation. Consider a  $m \times n$  torus which is viewed as a rectangular array (with opposite sides joined as well) with  $n$  processors per row and  $m$  processors per column. Let  $R = \lceil \frac{n-1}{2} \rceil$ ,  $L = \lfloor \frac{n-1}{2} \rfloor$ ,  $A = \lceil \frac{m-1}{2} \rceil$ , and  $B = \lfloor \frac{m-1}{2} \rfloor$ .

##### *Broadcast of New System Tops*

The framework for a broadcast will then consist of the following three parts:

- B1) A transputer that **finds** a new system top sends this new value to all of its four neighbours along with a forwarding distance (i.e. how many processors further the information needs to be sent) of  $R$  to the "right" and  $L$  to the "left",  $A$  "above" and  $B$  "below".
- B2) A new top message with forwarding distance greater than one **received** from the left (right, above, below) will always be forwarded right (left, below, above resp.) with forwarding distance decremented by one.
- B3) A transputer that **receives** a new system top from either the left or right will always forward it both above and below with forwarding distance of  $A$  and  $B$  respectively.

##### *Request for New Bodies*

When it comes to requests for new bodies it was decided that forwarding this type of communication needlessly complicates the logic required. Therefore whenever a transputer requires more work, it asks one of its immediate four neighbours for some bodies. If the first request fails (i.e. the neighbour has insufficient bodies to give up and answers with a "no"-tag instead of a mini-system of bodies) then another neighbour will be asked. This cycle will continue until a request is satisfied. Note that indirectly work to be shared moves further than the immediate nearest neighbours. If a cluster of processors need work, it will be brought in from the peripheral neighbours and then be available to move further inward. The framework for this communication is:

If conditions are such that a transputer requires more bodies.

- R0) Set  $N = 0$ .
- R1) A request is sent to neighbour  $N$ .
- R2) If the answer is negative then set  $N = (N + 1) \bmod 4$ , go to R1.

The answer from a transputer is simply:

- A1) The transputer that receives a request either shares some of its own system and responds with a mini-system of bodies or responds with a negative response.

### *Heuristics*

We consider heuristics (methods) for sharing the work. Should a transputer ask for more work only when it runs out of bodies or should it happen sooner? In that case: what conditions will trigger a request? And when responding to a request: How many bodies should be given away? And what kind of bodies (random or a subset of the biggest ones)? The performance of the algorithm will vary greatly depending on what methods are used. We explore the following possibilities:

#### *Conditions triggering a request for bodies*

- CT1) No bodies remain.
- CT2) The largest body left in the system is smaller than some predefined size.
- CT3) The largest body left is smaller than some dynamically alterable size (e.g. whenever the largest body is smaller than 10% of the largest of all bodies in the whole system).

#### *Creating the mini-system to share*

- MS1) The largest body only.
- MS2) Half of all bodies (up to some maximum number).
- MS3) Half of those bodies that are larger than some predefined size (up to some maximum number).
- MS4) Half of those bodies that are larger than some dynamically alterable size (see CT3 above) (again up to some maximum number).

The next section presents results for some of these cases. Those combinations not considered very useful are not shown.

## 5. SOME TEST RESULTS

An implementation in OCCAM-2 of the algorithm described above was written using a 4x8 torus of 32 transputers. This implementation was then used to find the most suitable combinations of above heuristics. All the testing was done with this particular function:

$$f(x) = -0.1 * \text{COS}(5\pi * x_1) + x_1^2 - 0.1 * \text{COS}(5\pi * x_2) + x_2^2$$

In all cases, the algorithm was terminated after a predetermined amount of time. Figures 5 to 11 summarize all the results.

Figure 5 shows the result of using the most primitive method for sharing work. This is the combination of CT1 and MS1 (see Section 4), asking for one new body whenever a transputer runs out of bodies and always responding to a request with the largest body that a requested transputer has left in its system. The graphs show how the size of the biggest body left in the system decreases with longer running times (runtime in seconds on x-axis). During the Get Next Point step of MB (see Section 2) one can use either the deepest (biggest) body left in the system or the deepest oldest body. It is clear from the graphs that the strategy of choosing the deepest is better. Therefore the results shown in the other figures are obtained using the "deepest"-strategy.

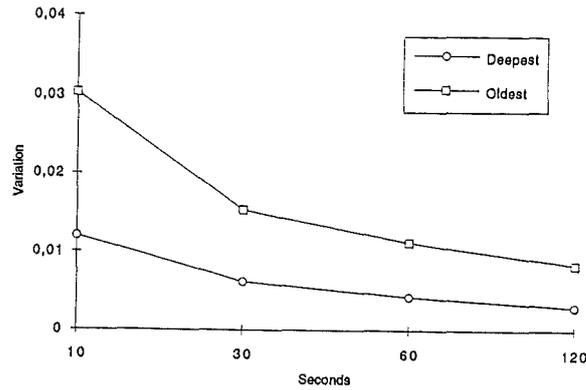


Figure 5. Variation when responding with only one body

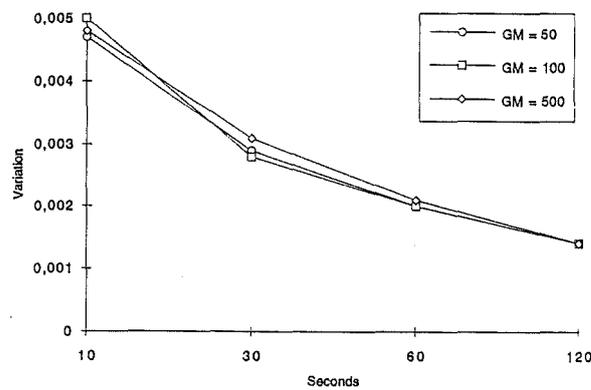


Figure 6. Variation when responding with more than one body

Figure 6 shows the effect of giving away more than just the largest body as an answer to a request. The reason for a maximum number on the amount of bodies that can be given away is due to limitations in the implementation language (no dynamic memory allocation in OCCAM-2). When responding to a query a transputer gives away half of its bodies but never more than the maximum number (GM in the figure). The effect of giving away a maximum of 50, 100 or 500 bodies on the size of the biggest body over time is shown. This method clearly reduces the amount of communication between the transputers and therefore produces better results compared to those shown in Figure 5.

One can actually see an order of magnitude improvement in the result (size of biggest body). When giving away a maximum of 100 or 500 bodies the size of the biggest body doesn't get reduced very much compared to a maximum of 50 bodies. Nevertheless some other benefits exist which can be seen in Figure 7.

It shows the ratio between the maximum number of iterations performed by any transputer and the average number of iterations performed by all transputers. One can see that it gets much closer to 1 when giving away more bodies. This shows that the work is now much more evenly distributed. Therefore more transputers can be kept working for a longer period between requesting new bodies (the more bodies a transputer gets, the longer it will take it to get rid of them). This results in better efficiency (more time spent on working and less on communicating/waiting). It therefore seems beneficial to give away as many bodies as possible compared to giving just a few. The test results shown in the remaining figures are obtained using 500 as the maximum number of bodies given as a response to a request (i.e. GM = 500).

Figure 8 shows the effect of giving away only "big" bodies when responding to a request. A constant limiting size (0.0005 in the figure) is used and when responding to a request only those bodies that are bigger than this limit are given away. Two graphs are shown, one giving the results when a transputer asks for more work only when it runs out of bodies (CT1), and one

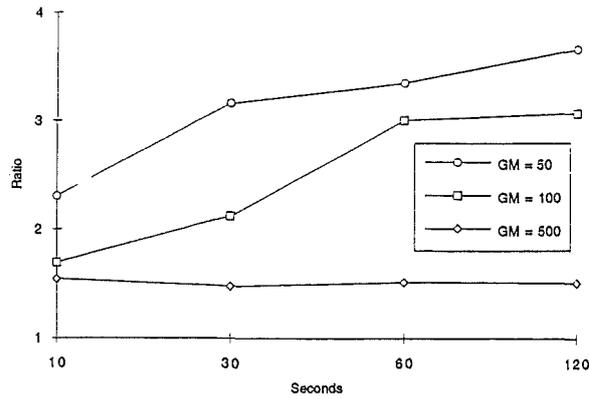


Figure 7. Ratio between maximum and average number of iterations from Figure 6

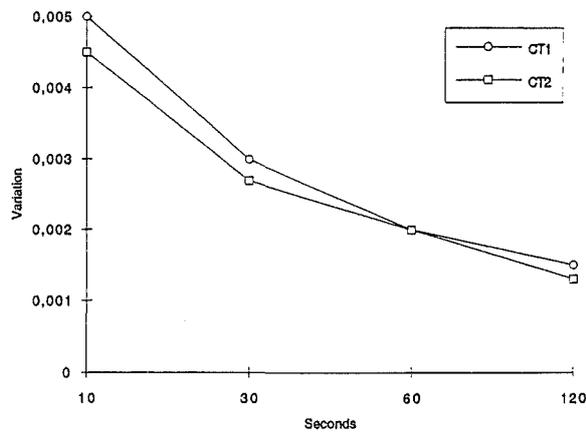


Figure 8. Variation when responding with bodies bigger than a constant limit

giving the results when a transputer asks for more work whenever its biggest body is smaller than the limit (CT2).

This method is beneficial in the sense that it forces the transputers to work on all the biggest bodies first, postponing work on small bodies. This means that the program will now converge towards the limiting size much faster than if the limit was not defined.

We can also see that introducing a limit has further evened out the differences between the transputers (Figure 9). The difference between the maximum number of iterations and the mean number of iterations per transputer is now much smaller than in Figure 7.

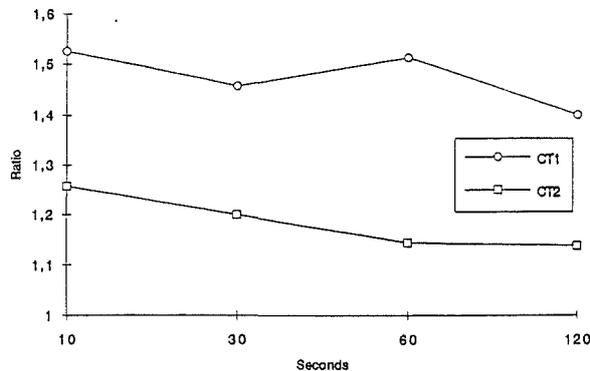


Figure 9. Ratio between maximum and average number of iterations from Figure 8

The drawback of a constant limiting size is that once the limit is reached no further progress can be made. The limit therefore becomes a termination criteria which prevents the program from ever reaching a "final" (global) solution. Therefore a dynamic strategy is needed.

To be able to retain the benefits of a limit (faster convergence towards the limit) and still be able to continue working until a global solution is found one must introduce a dynamic limit instead of a fixed one. This is implemented in such a way that every 100 iterations a transputer broadcasts the size of its biggest body (this broadcast is handled the same way as the broadcast of a new system top). In this way every transputer has at least a rough idea of the largest body of all. Figure 10 shows the effect of reducing the limiting size as the size of the biggest body in

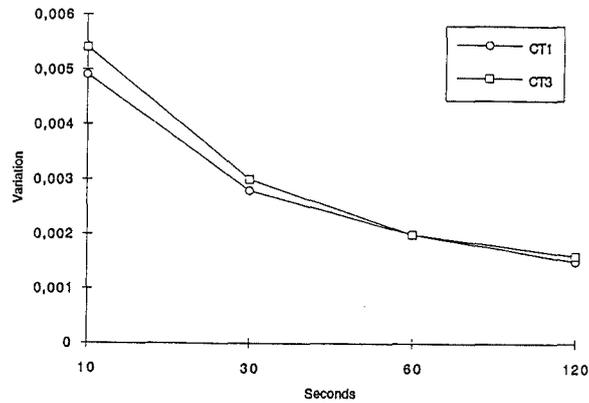


Figure 10. Variation when responding with "big" bodies

the system reduces. When responding to a request only those bodies bigger than 10 percent of the current limit will be given away.

Again two graphs are shown. One (CT1) gives the results when a transputer asks for more work only when it runs out of bodies. The other (CT3) gives the results when a transputer asks for more work whenever its biggest body is smaller than 10 percent of the current limit (approximately known from the regular broadcasts). Figure 11 shows the ratio graphs for this method.

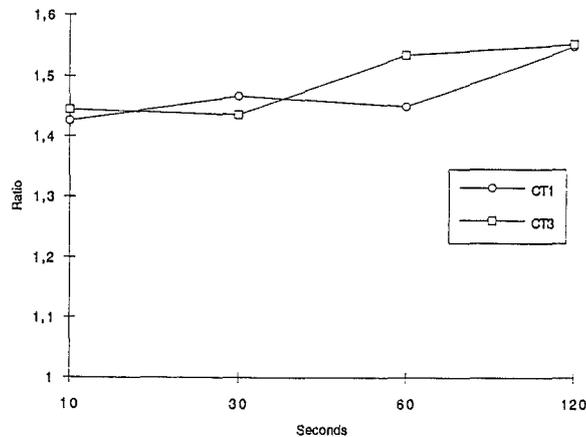


Figure 11. Ratio between maximum and average number of iteration from Figure 10

As can be seen from the last two figures no further improvements were made by introducing the concept of dynamically alterable limits. One reason for this is probably the added work needed to broadcast the biggest bodies every 100 iterations and the added logic that take care of this. Nevertheless we believe that this method is the best. It has two benefits:

1. It retains the even spread of work, most transputers are working on the big bodies and work on smaller bodies is postponed until later (this is good, based upon the assumption that a big body can plunge the system top by a larger amount than a small body).
2. The fixed limit introduces a horizon that cannot be passed but a dynamic limit results in a horizon that is constantly being pushed forward, thereby allowing work to continue.

## 6. CONCLUSION AND FUTURE WORK

This paper explored a parallel implementation of multidimensional bisection using transputers. Many different ways to realize the inter-transputer communication were examined. Also a set of heuristics were tried out in order to find out the best way to divide the system of bodies among available transputers. As a result we conclude that the most important thing in order to achieve fast convergence towards the minimum is the even distribution of "large" bodies. This seems justified also by the fact that cutting/capping a body can only remove bodies that are smaller than (or the same size as) the current body. Therefore working on small bodies will not remove the big ones, it can only reduce their size. This means that faster convergence towards the minimum is achieved by working on the biggest bodies first.

Using a 32 transputer configuration speed-ups of 6 were achieved compared to a sequential version. This gives an efficiency of only about 0.2. Nevertheless, as the time to compute one function evaluation increases the efficiency should improve. This because a larger fraction of the runtime would then be spent on calculating function evaluations and less and less on communicating.

Future research include looking at more test cases and optimizing values for the parameters mentioned in Section 5. These include how often to broadcast size of biggest body, when to request new bodies, and how many and what kind of bodies to respond with.

## REFERENCES

1. W. Baritompä, Customizing methods for global optimization - a geometric viewpoint. *J. Global Optimization* **3** no **2**, 193-212 (1990).
2. W. Baritompä, Multidimensional Bisection: A Dual Viewpoint, *Research Report No. 93 Dept of Mathematics University of Canterbury* (1993) *Computers and Mathematics with Applications* (to appear).
3. R.H. Mladineo, An algorithm for finding the global maximum of a multimodal, multivariate function, *Mathematical Programming*, **34** 188-200 (1986).
4. S.A. Piyavskii, An algorithm for finding the absolute extremum of a function, *USSR Computational Mathematics and Mathematical Physics* **12**, 57-67 (1972).
5. B.O. Shubert, A sequential method seeking the global maximum of a function, *SIAM J. Numerical Analysis* **9**, 379-388 (1972).
6. G.R. Wood, Multidimensional bisection and global optimization, *Computers and Mathematics with Applications* **21**, 161-172 (1991).
7. G.R. Wood, The bisection method in higher dimensions, *Mathematical Programming*, **55** 319-337 (1992).
8. Zhang Baoping, G.R Wood and W. Baritompä, Multidimensional Bisection: the performance and the context, *J. Global Optimization* **3** no **3**, 337-358 (1992).