Numerical Optimisation Centre

TECHNICAL REPORT NO. 127 MAY 1982

THE IMPLEMENTATION OF A PARALLEL VERSION OF PRICE'S (CRS) ALGORITHM ON AN ICL DAP

by

Paul G. Ducksbury

The Hatfield Polytechnic

PAUL DUCKSBURY

THE HATFIELD POLYTECHNIC

NUMERICAL OPTIMISATION CENTRE

THE IMPLEMENTATION OF A PARALLEL VERSION OF PRICE'S (CRS) ALGORITHM ON AN ICL DAP

by

Paul G. Ducksbury

TECHNICAL REPORT NO. 127

MAY 1982

Abstract

This report describes the way in which a parallel version of Price's (controlled random search) algorithm was implemented on an ICL Distributed Array Processor (the DAP situated at Queen Mary College, London) for the solution of multi-extremal global optimisation problems; together with the corresponding results that were obtained from it.

The author would like to thank the SERC for their support and the Queen Mary College DAP Support Unit for permission to use their system.

1. The Problem

The problem of global optimisation is as follows:

$$\min F(x) \qquad x \in S \subset R^n$$

where $S = \{x: a_i \leq x_i \leq b_i\}$

Assuming that the function $F(x)$ is non-convex and has more than one local minimum, then the problem is to isolate the point x^* with the least function value (which is by definition, the global minimum).

An international study comparing the relative efficiency of a number of programs for the solution of the global optimisation problem is reported in Dixon & Szegö [5]. That study indicated that most of the efficient codes are based on probabilistic principles and that the heuristic CRS algorithm proposed by Price [6] was one of the most successful. He has recently proposed a simplified version [7] and this was chosen for implementation on the parallel system.

The probabilistic method of solution relies on the following assumption:

if we have a finite number of points chosen at random and uniformly distributed in the region S, then if A is a subset of S with a measure m such that

$$\frac{m(A)}{m(S)} \geq \alpha > 0$$

and p(A,N) being the probability that at least one point of a sequence of N random points lies in A,

then $\lim_{N \to \infty} p(A,N) = 1$

The heuristic method then aims at improving the best points located in the random search.

2. System for the implementation

The ICL-DAP is a single instruction multiple data machine (SIMD) having a total of 4096 individual processors (arranged in a 64 x 64 matrix) each one having its own 1 bit full adder, 1 bit accumulator, a carry register and what is known as an activity switch used for "switching off" a particular processor if need be.

The total store of the machine is at present 4096 bits per processor, and a typical single length matrix requires 32 bits, one bit per plane, allowing approximately 120 matrices of length 32 bits to be held. The DAP is connected to an ICL 2980 host computer at Queen Mary College.

3. Language for the implementation

The language used on the DAP is the DAP-Fortran language which is an extension to the existing ICL 2900 Fortran, the extension being a set of 51 built-in aggregate functions for manipulation of vector and matrix values. Details of the languages and the supporting system can be found in technical publications [1, 2, and 3].

4. Price's algorithm for a DAP

The sequential algorithm of Price has been previously modified for running on an ICL-DAP for small values of n, as is described in Dixon & Patel [4]. The main features of the modified Price's algorithm are contained on the flow chart (Fig. 1).

Briefly what happens is this, 3n x 4096 points are generated over the domain S and their function values determined. (3n points allocated to each processor) the largest valued n points per processor are then rejected and the overall 4096 remaining largest values are chosen, one being placed in 'each' processor.

In order to generate new trial points P_i we choose 4096*n points at random (n points coming from each of the processors) from the remaining

set of 2n*4096 and together with the minimum poing f_L, determine the centroid followed by the new trial point matrix P.

i.e.

Suppose we have chosen n points R_2, R_3, ..., R_{n+1} at random and R_1 = L then

$$\underline{G} = \left(\sum_{i=1}^{n} \underline{R}_i\right)/n$$

and

$$\underline{P} = 2\underline{G} - \underline{R}_{n+1}$$

If the new points generated are within the domain S then we evaluate F(P) otherwise we must repeat by choosing more points at random, for those which are not within S, we also repeat from this stage if F(P) is worse than the existing point.

At this stage (after one or possibly more iterations) we have a full set of 4096 points which are contained within the domain S and whose function values are less than the corresponding maximum function values. It is these improved values that are used in the replacement process and the test for termination is carried out.

The termination criterion examines the 3 smallest values of the function value L1, L2, L3 such that L1 < L2 < L3 then if $|L1 - L2| <$ and $|L2 - L3| <$ and $|L2 - L3| < \epsilon$ (ϵ = some tolerance) we stop, otherwise repeat by selecting the 4096 new largest values etc.

The algorithm also contains as a safety measure an upper bound on the number of iterations (in this case 40 - purely an arbitrary figure).

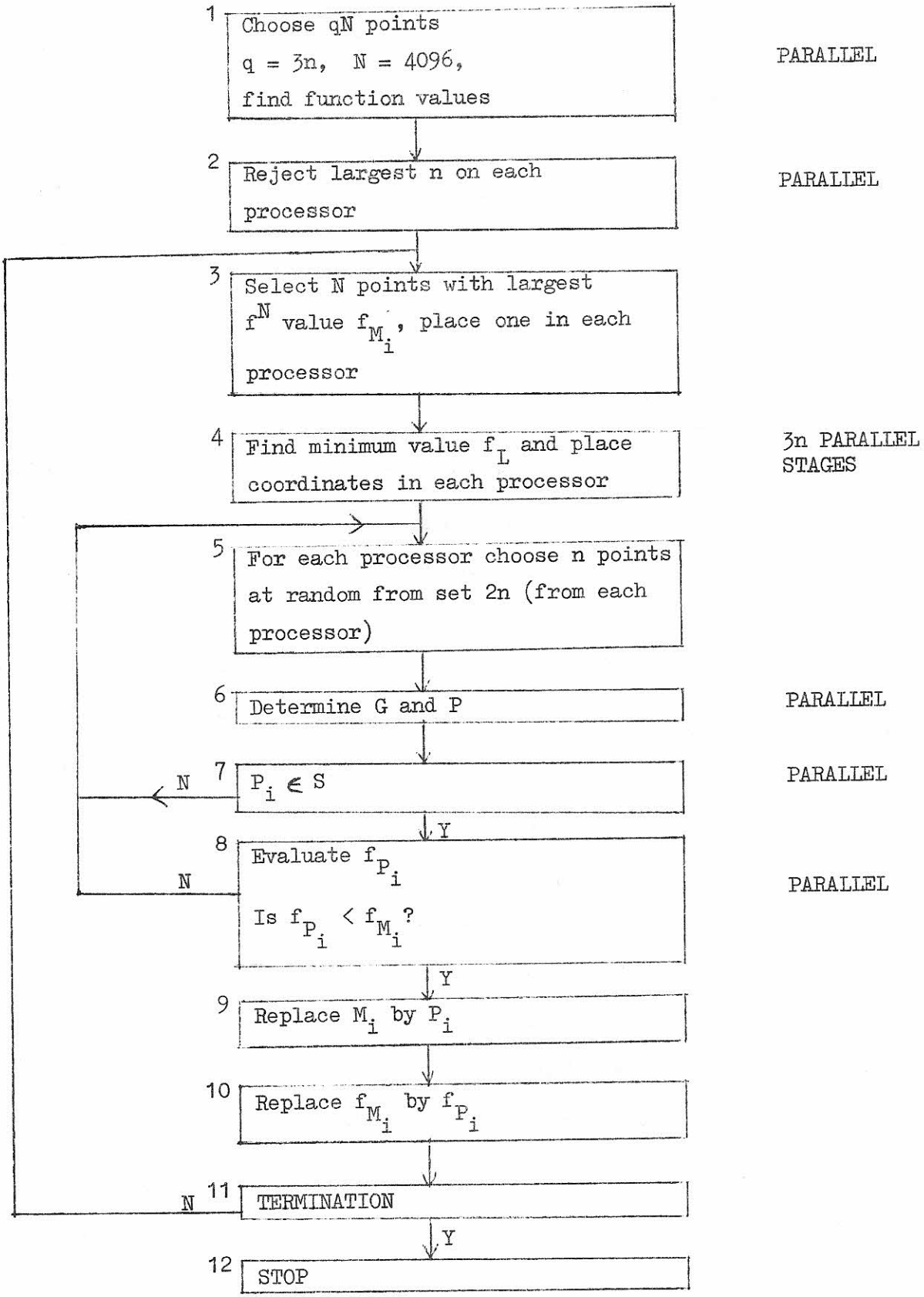
1
Choose qN points
q = 3n, N = 4096,
find function values
PARALLEL
2
Reject largest n on each processor
PARALLEL
3
Select N points with largest f^N value f_{M_i}, place one in each processor
4
Find minimum value f_L and place coordinates in each processor
3n PARALLEL STAGES
5
For each processor choose n points at random from set 2n (from each processor)
6
Determine G and P
PARALLEL
7
$P_i \in S$
N
PARALLEL
Y
8
Evaluate f_{P_i}
Is $f_{P_i} < f_{M_i}$?
N
PARALLEL
Y
9
Replace M_i by P_i
10
Replace f_{M_i} by f_{P_i}
11
TERMINATION
N
Y
12
STOP

Figure 1

5. Initial results

The initial results obtained from this program were rather disappointing when compared with the sequential algorithm (the sequential one being run on a DEC system 10, 1091 here at Hatfield) as can be seen from Table 1 (sequential vs. 'old' parallel). It was the case that three sections in the program had needed to be coded sequentially and these sections were pinpointed as being the main time consumers. The available diagnostic facilities on the DAP allow for the timings of individual sections of code (or even individual statements) to be recorded. They were as follows:

(i) Step 3 on Figure 1, when choosing the overall 4096 largest function values it was necessary to code it sequentially, namely, find the largest value, mask it out, then find the next largest, etc.

(ii) Step 5 on Figure 1, when choosing n points at random for each processor it proved to be extremely difficult to find a method for doing it in parallel, since it was required to choose n points at random, avoiding the points previously rejected and keeping track of where the chosen points had come from (eventually the coordinates were recorded namely, row, column, plane).

(iii) Steps 9 and 10 on Figure 1, these two sections formed the third major sequential point in the program. When replacing the points M_i by new points P_i and similarly f_{M_i} by f_{P_i}, they had to be replaced in the positions from which they initially came.

These three points in the program were modified by means of changes to the algorithm, resulting in the new version of the parallel algorithm to be described in Section 6.

6. The new Price's algorithm for a DAP

The algorithm described in Section 4 was modified to remove the three main sequential sections of code. The main details of the new algorithm are shown in Figure 2 but briefly they are as listed below:

(i) After rejecting the largest n points on each processor we 'compact down' the matrices MI and FMI (holding the coordinates and function values of points respectively) by removing the rejected values, this means we no longer need to take care to avoid them.

(ii) Instead of choosing the overall 4096 maximum values, we now select 4096, but the maximum from each processor, which can be done quite simply in parallel.

(iii) Instead of choosing n points at random, one from each processor, we now choose n matrices at random. This means that it is now no longer necessary to record the positions of the randomly selected points for later reference, since they have all come from one of n matrices.

(iv) Since we are now choosing whole matrices at random rather than individual points, then the replacement of old points by new ones is a lot simpler, i.e. we can generate a logical mask, containing TRUE values, only at positions where the new point is contained within the domain S AND the function value f_{P_i} is less than f_{M_i}. This mask then being used in a series of simple assignments.

(v) When we compacted down the two matrices there was space left over, n planes in FMI, n^2 in MI, this space is now used for holding the centroids, new trial points, maximum values and the function values of new trial points.

(vi) The random number initialisation routine and generator were also changed to faster but adequate routines.

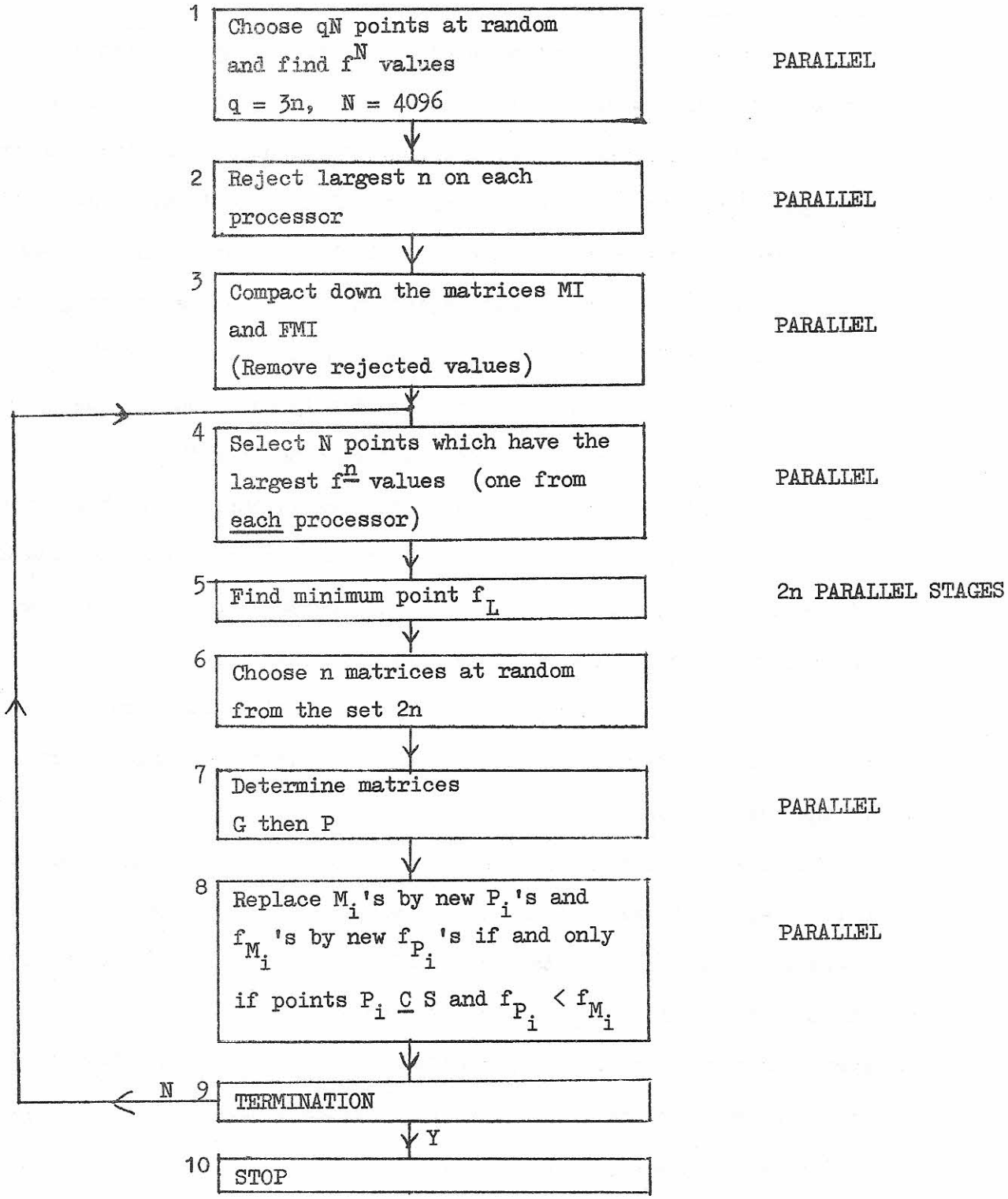
1
Choose qN points at random and find f^N values
q = 3n, N = 4096
PARALLEL
2
Reject largest n on each processor
PARALLEL
3
Compact down the matrices MI and FMI
(Remove rejected values)
PARALLEL
4
Select N points which have the largest f^n values (one from each processor)
PARALLEL
5
Find minimum point f_L
2n PARALLEL STAGES
6
Choose n matrices at random from the set 2n
7
Determine matrices G then P
PARALLEL
8
Replace M_i's by new P_i's and f_{M_i}'s by new f_{P_i}'s if and only if points $P_i \subset S$ and $f_{P_i} < f_{M_i}$
PARALLEL
N 9
TERMINATION
Y
10
STOP

Figure 2

7. Final results

The results of this new version of the algorithm are also displayed in Table 1 (sequential vs. 'new' parallel). This time a positive improvement can be seen compared with the sequential algorithm and an even greater improvement on the 'old' parallel algorithm.

A direct comparison of the times for the sequential and the new parallel yields the improvement factor which can be seen to range from 3.2 up to 68. This considerable range can be explained by the fact that the lower improvement factors came from functions which had just one global minimum, whereas the larger factors came from the functions possessing multiple global minima. The best example is probably the SHUBERT function which has 18 global minima; the 'new' parallel algorithm picked up 11 of these minima in one run, whereas the sequential algorithm required 10 runs to pick up these same 11 minima.

It can also be seen that the number of parallel function evaluations needed has been drastically reduced in most cases to single figures but at the most to just 14.

Table 2 contains the numeric results of the programs and Table 3 the actual values (obtained from [5] and elsewhere). It can be seen that there is a reasonable comparison in all of the test cases except for SHEKEL's function where the algorithm only managed to locate the approximate position of the global minima, this was then allowed to run for 40 iterations and also 8 different random sequences but still failed to improve upon the quoted figure.

Taking the percentage differences of the values we find that the results for the sequential algorithm were within an average of 0.65% of the actual values and for the new parallel algorithm within an average of 1.54% but if we include the results from the SHEKEL function then this figure worsens to 4.57%.

8. Conclusions

Despite the fact that the numeric results for the new algorithm were very slightly worse than those of the sequential algorithm ($4.57 - 0.65 \approx 3.9\%$) there must be an advantage in a considerable reduction in the number of parallel function evaluations carried out (particularly on functions for which a large proportion of the time is spent in the function evaluation routine) thus leading to a reduced time.

The CRS optimisation method described in [5] on which this method is based is designed for thoroughness of the search rather than speed of convergence and where possible the global minima discovered could be refined using faster methods.

The algorithm is only suitable for small values of n, namely those in the range

$$2 \leq n \leq 5$$

The restriction on larger values is one of the available storage space. A possible alternative for larger dimensional problems would perhaps be to reduce the number of points generated to say 2n per processor but this would later limit the choice of the matrices in a random way.

The program is now successfully running on an ICL DAP. The two versions of the implementation illustrate an interesting property, namely, the algorithm (or any algorithm) must be designed with the available facilities in mind, not necessarily the detailed hardware facilities but rather the software ones, i.e. the built-in DAP functions and the best way in which to use them.

Hence no matter how efficient the machine is, a number of sequential code sections (as is the case with the initial implementation) in the program can totally destroy any possible advantage which comes from using the parallel machine.

FUNCTION	SEQUENTIAL		NEW PARALLEL		OLD PARALLEL		IMPROVEMENT
	Time (s)	f^N eval.	Time (s)	f^N eval.	Time (s)	f^N eval.	Note (3)
SIX HUMP CAMEL BACK (2 global minima)	0.36 0.81 1.17	288 798 1086	0.093	7 Note (1)	95.6	21	x 12.5
BRANIN (3 global minima)	0.47 0.97 1.73 3.17	360 654 1024 2038	0.118	8 Note (1)	25.7 25.8 Note (2)	9 9	x 26.8
GOLDSTEIN & PRICE (1 global minimum)	0.45	481	0.13	8	37.78	12	x 3.5
HARTMANS N = 3 (1 global minimum)	0.88	503	0.276	11	67.33	13	x 3.2
N = 6			INSUFFICIENT STORE				
SHEKEL (1 global minimum)							
m = 5 Note (5)	3.89	2732	0.453	14	NOT RUN		x 8.6
m = 7 Note (5)	3.82	2423	0.55	14	NOT RUN		x 6.9
m = 10 Note (5)	4.31	2567	0.693	14	NOT RUN		x 6.2
SHUBERT 2-D (18 global minima)	average 1.58/min total 15.8 Note (4)	average 995	0.233 Note (4)	8	NOT RUN		x 68

TABLE 1

NOTES

(1) Multiple global minima were identified in one run
(2) Two runs were needed to locate 3 global minima
(3) Improvement of new parallel as compared with the sequential
(4) The parallel algorithm picked up 11 minima in one run hence the same 11 minima from the sequential version were used for the comparison (these 11 being picked up in 10 runs)
(5) For this particular function the approximate location only was located for the global minimum.

FUNCTION	SEQUENTIAL	NEW PARALLEL	OLD PARALLEL
SIX HUMP CAMEL BACK (2 global minima)	f = -1.0312 @ 0.09047, -0.70548 & -0.08851, 0.71195	f = -1 030088 @ 0.09524, -0.7262 & -0.07299, 0.71892	f = -1.03035 @ -0.07846, 0.72161 Note (1)
BRANIN (3 global minima)	f = 0.39797 @ 3.14294, 2.2821 & 9.43796, 2.4653 & -3.1364, 12.2577	f = 0.401413 @ 3.16611, 2.23063 & 9.41012, 2.5422 & -3.12068, 12.12934	f = 0.40254 @ 3.1431, 2.272 & 9.3941, 2.477 & -3.16734,12.375
GOLDSTEIN & PRICE (1 global minima)	f = 3.00008 @ 0.00061, -0.99986	f = 3.041037 @ 0.01146, -1.00212	f = 3.037369 @ 0.010315, -0.99176
HARTMANS N = 3 (1 global minima)	f = -3.86223 @ 0.12838, 0.55814, 0.85407	f = -3.85239 @ 0.12173, 0.57061, 0.85807	f = -3.85532 @ 0.1454, 0.5416, 0.85097
SHEKEL N = 4 (1 global min) m = 5	f = -10.15269 @ 3.99941, 3.99815, 4.00085, 3.99975	f = -4.85933 @ 4.01411, 3.96868, 3.74164, 4.21136	NOT RUN
m = 7	f = -10.4028 @ 4.00118, 3.99993, 3.99974, 4.00025	f = -5.106936 @ 4.01411, 3.96868, 3.74164, 4.21136	NOT RUN
m = 10	f = -10.5362 @ 4.00182, 3.99978, 3.99978, 3.99994	f = -5.2288 @ 4.01411, 3.96868 3.74164, 4.21136	NOT RUN
SHUBERT 2-D	f = -186.38818 @ -7.0817, -7.7046 -1.4555, -0.80274 4.8788, -7.06644 -0 785, 4.8523 -7.7083, -7.0835 4.8663, -0.7934 -1.42505, 5.4828 -7.7083, -0.8001 -7.7077, 5.483 -0.802, -7.7052 5.4828, 4.8579	f = -186.5548 -7.09226, -7.7066 -1.40794, -0.80522 4.88029, -7 0847 -0.79988, 4.89147 -7.66859, -7.0571 4.84246, -0.8475 -1.4149, 5.42207 -7.76933, -0.79236 -7.66386, 5.43725 -0.80595, -7.64545 5.41464, 4.81433	NOT RUN

TABLE 2

NOTE

(1) Only one of the global minima was located, the random sequence was changed several times

FUNCTION	ACTUAL VALUES
SIX HUMP CAMEL BACK	f = -1.0316285 @ 0.08983, -0.7126 and -0.08983, 0.7126
BRANIN	f = 0.397887 @ 3.14159, 2.275 and 9.42478, 2.475 and -3.14159, 12.275
GOLDSTEIN & PRICE	f = 3.0 @ 0.0, -1.0
HARTMANS N = 3	f = -3.86278 @ 0.11484, 0.555515, 0.852551
SHEKEL N = 4 m = 3	f = -10.1532 @ 4.0003, 4.00016, 4.00002, 4.00011
m = 7	f = -10.4029 @ 4.00052, 4.00078, 3.99943, 3.99957
m = 10	f = -10.5364 @ 4.00066, 4.00054, 3.99955, 3.99948
SHUBERT 2-D	f = -186.73091 @ -7.0835, -7.70831 -1.42513, -0.80032 4.85805, -7.0835 -0.80032, 4.85805 -7.70831, -7.0835 4.85805, -0.80032 -1.42513, 5.48286 -7.70831, -0.80032 -7.70831, 5.48286 -0.80032, -7.70831 5.48286, 4.85805

TABLE 3

9. References

1. "DAP introduction to Fortran programming," Technical Publication TP 6755, ICL Putney, London SW15.

2. "DAP-Fortran Language," Technical Publication TP 6918, ICL Putney, London SW15.

3. "DAP - Developing DAP programs," Technical Publication TP 6920, ICL Putney, London SW15.

4. Dixon, L.C.W. and Patel, K.D., "The place of parallel computation in numerical optimisation II, the multi-extremal global optimisation problem," Technical Report 119, Numerical Optimisation Centre, The Hatfield Polytechnic.

5. "Towards global optimisation 2," edited by L.C.W. Dixon and G.P. Szegö, North Holland Press, 1978.

6. Price, W.L., "A heuristic CRS algorithm," in Towards global optimisation 2, edited by L.C.W. Dixon and G.P. Szegö, North Holland Press, 1978.

7. Price, W.L., "A new version of the controlled random search procedure for global optimisation," Technical Report, Engineering Dept., University of Leicester, 1981.