ORiON, Vol. 5, No. 2, pp. 75-83 *ISSN 0259-191 X*

EVALUATING PARALLEL OPTIMISATION ON TRANSPUTERS

A G Chalmers, J W Hearne and C J Scogings
Department of Computer Science
and
Department of Mathematics & Applied Mathematics
University of Natal
P O Box 375
PIETERMARITZBURG 3200
South Africa

ABSTRACT

The faster processing power of modern computers and the development of efficient algorithms have made it possible for operations researchers to tackle a much wider range of problems than ever before. Further improvements in processing speed can be achieved utilising relatively inexpensive transputers to process components of an algorithm in parallel.

The Davidon-Fletcher-Powell method is one of the most successful and widely used optimisation algorithms for unconstrained problems. This paper examines the algorithm and identifies the components that can be processed in parallel. The results of some experiments with these components are presented which indicates under what conditions parallel processing with an inexpensive configuration is likely to be faster than the traditional sequential implementations. The performance of the whole algorithm with its parallel components is then compared with the original sequential algorithm.

The implementation serves to illustrate the practicalities of speeding up typical OR algorithms in terms of difficulty, effort and cost. The results give an indication of the savings in time a given parallel implementation can be expected to yield.

INTRODUCTION

The tools of the operations researcher have improved considerably over the last decade or so. Jobs that once required so much computational time from a mainframe computer that they typically had to be run over weekends can now be done inter-actively on a personal computer (PC). These advances have come about as a result of improvements in the efficiency of the algorithms used by the operations researchers as well as major increases in the speed and power of computers. Despite the advances there is still a great need for faster processing power, but physical laws are placing constraints on possible speed enhancements of the single processor. Faster computing ability can be achieved by Vector Processors, but their price puts them well beyond the reach of the average operations researcher. However, a reasonably priced parallel system can be constructed with transputers [1].

When transputers first came on sale three years ago they were hailed as a dream come true for scientists and engineers. However, with some exceptions in computer graphics, solid-state physics and fluid dynamics there has been very little exploitation of this new technology (Galea [2]). In particular, other than some experiments in simulation (Karplus and Shibata [3]) the operations research community appear to have ignored the advent of the transputer. There has been some work in the area of parallel programming on a variety of other machines (in fact the machines are usually not mentioned or are very expensive equipment). Byrd, Schabel & Schultz [4] and Grandinetti & Conforti [5] are typical examples of papers on parallel optimisation methods. Grandinetti & Conforti make use of a CRAY X-MP/48 supercomputer which is not available to most researchers.

The transputer, which costs only a little more than a PC, is a "complete computer" which can be used as a programmable component to construct a large multiprocessor system. Communication and synchronisation between the transputers are achieved via high speed serial links. It has been stated in some quarters that it is not worth studying transputer applications as they are an interim measure which will give way to more advanced parallel systems in the near future. On the contrary, the move is away from bit-serial array processors (such as the connection machine) towards parallel nodes which are stand alone processors, such as the transputer. Intel are even entering the market with their transputer look-alike, code named iWARP, due out towards the end of 1990 [6]. One of the major advantages of the transputer is that it is designed as a building block for parallel processors [7] and thus a small initial system can be gradually expanded into a larger parallel system as finances become available.

A typical distributed algorithm consists of a Collater process and one or more Worker processes. The Collater is responsible for apportioning the calculating effort to the Workers and collating and displaying the results as sent back. The Workers perform the required calculations on the data supplied by the Collater returning any results along with requests for more work.

The language Occam 2 was developed alongside the transputer to facilitate the distribution of an algorithm to several processes running on the individual processors. It is a small succinct language whose very foundation encompasses the concept of a process.

Other programming languages, such as Fortran, C and Pascal are also available on the transputer. These languages run within an occam harness which expedites the communication.

For this paper a system comprising three T800 transputers was configured as shown in figure 1. This is a minimal parallel configuration but suffices to illustrate the potential of parallel processing. Such a system costs about the same as three stand-alone PC's. In the next section an algorithm, well known to operation researchers, is examined with a view to implementation on such a parallel system. This is achieved by identifying those components of the algorithm that are suitable for parallel processing. Time tests are performed and compared with a traditional sequential implementation.

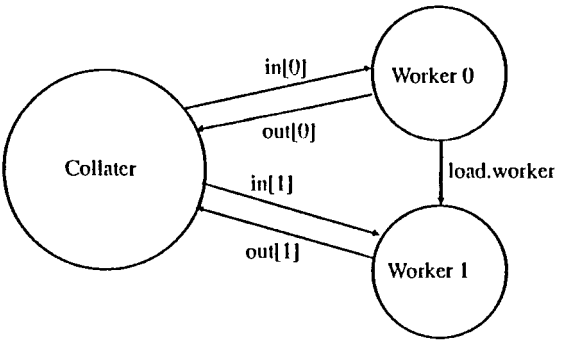

Figure 1: Configuration utilised

AN OPTIMISATION ALGORITHM

One of the most successful general purpose methods of optimisation for unconstrained problems is based on a method developed by Davidon [8] and extended by Fletcher and Powell [9] to become commonly called the Davidon-Fletcher-Powell (DFP) method. The method finds the minimizer of the function $f(\underline{x})$. Denoting the gradient of f at $\underline{x}$ by $g(\underline{x})$, the method proceeds as follows:

Step 1: Start with an initial estimate $\underline{x}_0$ and any positive definite matrix H_0, usually the identity matrix.

For i = 0, 1, 2 ... until satisfied do :

Step 2: * Set a direction of search $\underline{d}_i = - H_i \, g_i(\underline{x}_i)$

Step 3: * Perform a line search along $\underline{x}_i + \lambda \underline{d}_i$ to find λ_i which minimises $f(\underline{x}_i + \lambda_i \underline{d}_i)$

Step 4: $\underline{v}_i = \lambda_i \underline{d}_i$

Step 5: $\underline{x}_{i+1} = \underline{x}_i + \underline{v}_i$

Step 6: * Evaluate $f(\underline{x}_{i+1})$ and $g(\underline{x}_{i+1})$ and terminate the procedure if $| g(\underline{x}_{i+1}) |$ or $| \underline{v}_i |$ are sufficiently small.

Step 7: Let $\underline{u}_i = g(\underline{x}_{i+1}) - g(\underline{x}_i)$

Step 8: * Update matrix H

$$H_{i+1} = H_i + A_i + B_i$$

where $$A_i = \underline{v}_i \underline{v}_i^T \, / \left(\underline{v}_i^T \underline{u}_i \right)$$

$$B_i = H_i \underline{u}_i \underline{u}_i^T H_i \, / \left(\underline{u}_i^T H_i \underline{u}_i \right)$$

The line minimisation algorithm used in step 3 was a simple one-dimensional bracketing procedure to broadly bracket the minimum, followed by cubic interpolation to establish this minimum to a specific accuracy.

The steps marked with an asterisk ('*') indicate where parallelism was introduced into the algorithm. Steps 2 and 8 involve matrix multiplication while steps 3 and 6 involve function evaluations. Note that the evaluation of the gradient usually needs to be done numerically. This means that an objective function of n variables will need to be evaluated at least n+1 times to obtain a reasonable approximation of the gradient. In the next section the potential for time-saving by parallel processing in the two common operations of matrix multiplication and function evaluation is examined.

The experiments were done by implementing algorithms for these two operations in parallel on a configuration as shown in figure 1, and in sequence on a single T800 transputer. Both implementations were in Occam 2.

PARALLEL vs SEQUENTIAL : Matrix multiplication

A parallel matrix multiplication algorithm, eg A * B = C, utilises the independence of each row of a matrix in a processor farm topology as suggested by Packer [10] and May [11]. Each row of Matrix A is 'farmed out' to a Worker process where it is multiplied by a column of B, and the returned result stored in the appropriate row of matrix C. The communication speed of a T800 transputer link is 2.4 Mbytes per second overall rate for bi-directional data transfer [1]. For low dimension matrices this communication overhead resulted in better performance by the sequential method. However as the dimension of the matrices increased, so the parallel implementation easily outperformed the sequential one. This is illustrated in figure 2.

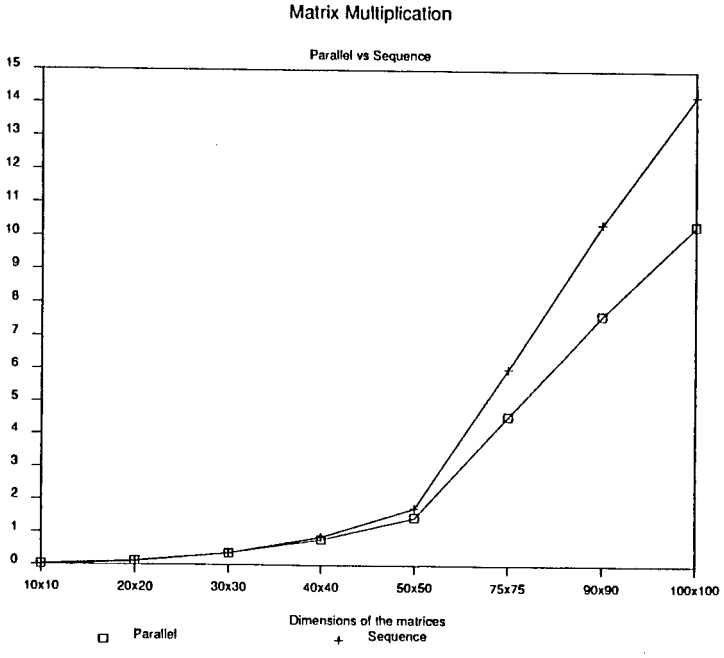

Figure 2: Time comparison of parallel and sequential matrix multiplication

PARALLEL vs SEQUENTIAL : Function evaluation

A function that takes one second to evaluate was used in this experiment. Figure 3 shows a comparison of parallel versus sequential execution of different numbers of function evaluations. When five evaluations of the function are required the parallel algorithm results in a saving of approximately 60% of the time taken for the sequential evaluations.

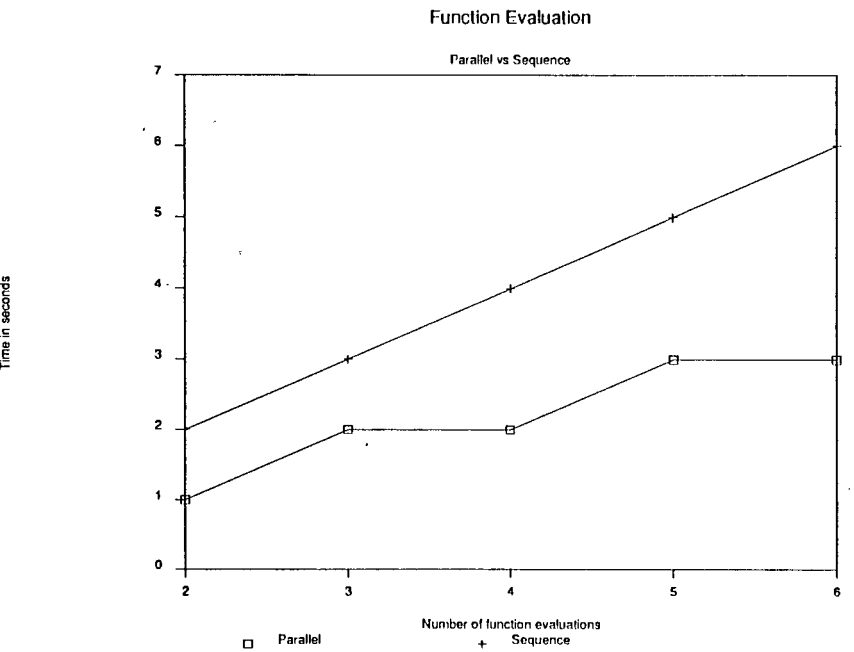

Figure 3: Time comparison of sequential and parallel function evaluations

It must be remembered that in many applications the evaluation of the objective function is very time consuming. For example, the objective function might be some performance index of a dynamic system. In such cases, a system of differential equations needs to be solved each time an evaluation of the objective function is required. Thus reducing the time taken for all function evaluations represents a significant improvement in overall performance of an optimisation package. Both techniques developed for the above tests were incorporated in the Davidon-Fletcher-Powell method.

PARALLELISM IN THE DFP METHOD

The farm topology as mentioned above was used throughout the DFP implementation. The most time consuming part was found to be step 3, the initial bracketing of the minimum prior to the application of the cubic interpolation. In the sequential implementation this meant the sequential evaluation of f at successive points along the line $\underline{x}_i + \lambda \underline{d}_i$ until the function stopped decreasing. The increment for the search was chosen as 0.1. In the parallel implementation, use was made of the two available Worker processes to perform "look ahead" calculations of the $f(\underline{x} + \lambda \underline{d})$'s, allowing two comparisons at each iteration instead of one. This contributed greatly in the time savings.

Two classical test functions were used to compare the performance of the DFP method in both the parallel and sequential implementations. These functions are popular when evaluating optimisation methods due to their difficulty in finding a minimum (Bunday and Garside [12]).

The first function optimised was the Rosenbrock function with 2 variables :

$$f(\underline{x}) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2 \qquad \min = (1,1)$$

with the customary initial values of (-1.2, 1).

The Powell function with 4 variables :

$$f(\underline{x}) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4 \qquad \min = (0,0,0,0)$$

with starting points (3, -1, 0, 1) was the second known function tested.

The third function optimised was an 8-variable function :

$$f(\underline{x}) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2 + 100(x_3^2 - x_4)^2 + (1 - x_3)^2$$
$$+ 100(x_5^2 - x_6)^2 + (1 - x_5)^2 + 100(x_7^2 - x_8)^2 + (1 - x_7)^2 \qquad \min = (1,1,1,1,1,1,1,1)$$

with starting points (-1.2, 1, -1.2, 1, -1.2, 1, -1.2, 1).

The times to optimise these three functions for the parallel and sequential implementations are shown in figure 4.

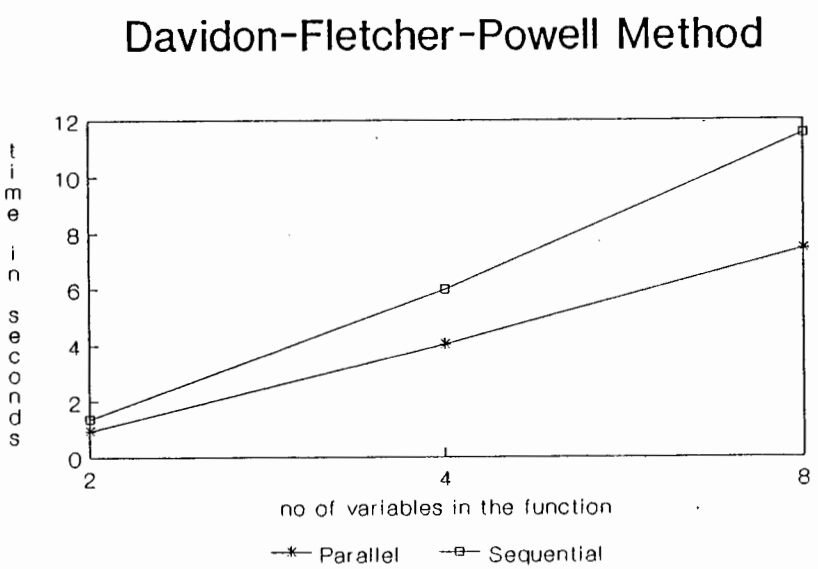

Figure 4: Performance evaluation of Davidon-Fletcher-Powell method

CONCLUSION

The advent of the transputer has put a parallel computer architecture within the financial means of most operation researchers. The Davidon-Fletcher-Powell method was implemented in both a sequential and parallel form in Occam 2 to evaluate the potential of the transputer for solving problems in operations research. Despite the limited equipment available (only three T800s), some very promising results were achieved. The time required to make one evaluation of either of the test functions is relatively short. Despite this, some significant savings of time were achieved in locating the optimum using the parallel algorithm. More dramatic results could be expected with more complex functions that require large amounts of time to evaluate.

It has been shown that it is desirable to perform such optimisations on parallel processors and that the transputer is particularly desirable because of the low cost and the ability to expand from a small working system. Although more transputers are not currently available, even greater improvements are expected when extra transputers are added to the existing system.

ACKNOWLEDGEMENTS

Thanks to Kevin Cameron and Nina Ligeti for helping with the programming.

REFERENCES

[1] **IMS T800 Architecture,** *INMOS Technical Note 16,* INMOS Bristol (1987).

[2] **E. Galea,** Supercomputers and the need for speed, *New Scientist,* 120:1638, 50-55 (12 November 1988).

[3] **W.J. Karplus & Y. Shibata,** The application of small peripheral array processors to the modelling of distributed parameter systems, *Simulation,* 46, 231-238 (1986).

[4] **R.H. Byrd, R.B. Schabel & G.A. Schultz,** Parallel quasi-Newton methods for unconstrained optimization, *Mathematical Programming,* 42, 273-306 (1988).

[5] **L. Grandinetti & D. Conforti,** Numerical comparisons of nonlinear programming algorithms on serial and vector processors using automatic differentiation, *Mathematical Programming,* 42, 375-389 (1988).

[6] *Electronics Weekly,* no 1456, Wednesday 12 April (1989).

[7] **R.W. Hockney & C.R. Jesshope,** *Parallel computers 2 : Architecture, programming and algorithms,* IOP Publishing Ltd., Bristol (1988).

[8] **W.C. Davidon,** Variable metric methods for minimization *AEC Research and Development Report,* ANL-5990, Argonne, Illinois, (1959).

[9] **R. Fletcher & M.J.D. Powell,** A rapidly convergent descent method for minimisation, *The Computer Journal,* Vol 6 No 2, 163-168 (1963).

[10] **J. Packer,** Exploiting Concurrency: A Ray Tracing Example, *INMOS Technical Note* 7, INMOS Bristol (1987).

[11] **D. May & R. Shepherd,** Communicating Process Computers, *INMOS Technical Note* 22, INMOS Bristol (1987).

[12] **B.D. Bunday & G.R. Garside,** *Optimisation methods in Pascal,* Edward Arnold Publishers Ltd, London (1987).