A Parallel Newton Method for Unconstrained Optimisation

Tim Oliver *
Centre for Mathematical Software Research
University of Liverpool

September 28, 1989

Abstract

This paper investigates the merits of message routing hardware (as proposed within the PUMA project [3]) for the implementation of Newton's method for unconstrained optimisation on a local memory MIMD Transputer-based architecture. The performance of such an architecture is predicted, and compared with the performance of current Transputer architectures.

1 Introduction

As new MIMD architectures are developed we are interested in the relative merits of these architectures for implementing parallel numerical methods. Important considerations are the computation rate of the individual processors and the speed of communication between processors. The communication cost is a combination of three factors: the raw speed at which data can be transmitted or received by a processor on a communication path, the number of communication paths that can operate concurrently on a processor, and the connectivity of processors by communication paths.

This paper considers two particular architectures based on the Transputer: the current static configuration of T8 Transputers, and the proposed message through-routing H1 Transputer architecture. We develop cost models for a particular algorithm on both of these architectures allowing the performances to be compared. The algorithm chosen for this purpose is Newton's method [5] for the unconstrained optimisation of a twice continuously differentiable function $F(x)$ of n real variables. Optimisation algorithms are difficult to parallelise because of the inherently sequential nature of the iterations whereby the $(k+1)$-th iteration cannot proceed until the minimum point of the k-th iteration has been calculated. However Newton's method does allow parallelism to be exploited within each iteration in the evaluation of the gradient vector g, the Hessian matrix G, and the calculation of the search direction s.

2 Algorithm

Each iteration within Newton's method consists of the following steps:

- Evaluate gradient vector g_k at x_k.
- Evaluate Hessian matrix G_k at x_k.
- Solve linear system for search direction s_k:

$$G_k s_k = -g_k \tag{1}$$

- Perform line search along s_k to find new estimate x_{k+1} of the minimum point of $F(x)$ in the direction s_k:

$$x_{k+1}: \qquad F(x_{k+1}) = \min_{\alpha} F(x_k + \alpha s_k) \tag{2}$$

*Many thanks to Len Freeman for his suggestions.

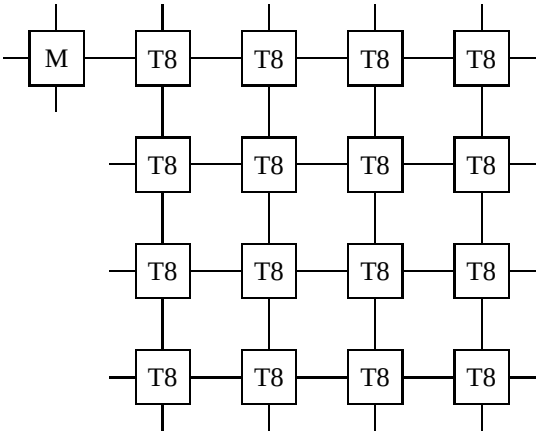

Figure 1: Grid configuration

The first three steps within each iteration are suitable for parallelisation. For optimum efficiency the distributed data structures used by these three parallel steps must be compatible. Without this provision there would be a requirement to redistribute the data between steps. For Newton's method the data structure is determined by the algorithm which solves the linear system to find the search direction. The algorithm chosen is row-oriented Gaussian elimination with partial pivoting. This parallel algorithm is known to perform well and has already been implemented on the current Transputer architecture (see [6] for details of the parallel algorithm). The distributed data structure therefore consists of scattered rows of G on each processor along with the associated elements of g. When the system of equations is solved the search direction s is also distributed, with each element of s located on the processor that holds the associated element of g. The other parallel steps, namely the evaluation of the Hessian matrix and gradient vector, must therefore produce a distributed data structure of scattered rows of G and the associated elements of g.

The line search step of each iteration is performed sequentially as it does not seem to have any scope for parallelism. This step must be preceded by collecting the distributed elements of s at one processor. Similarly, after the new minimum point x is found it must be broadcast to all the other processors before the next iteration can proceed.

We assume that a network of p processors is available, connected to a single master processor. For the current architecture we assume further that the processors are statically configured in a square grid topology (see Figure 1). The proposed architecture is not limited to a static configuration but allows dynamic interprocessor connectivity (see Section 4).

3 The User Interface

The user interface to the algorithm is through the user-provided functions to evaluate the gradient vector g and the Hessian matrix G. Sequential Newton algorithms expect the user to provide sequential functions that will calculate all of the Hessian and gradient. If this same user interface were retained for the parallel algorithm then it would not be possible to exploit parallel evaluation of the Hessian or gradient. However with only small changes to the user interface parallelism can be achieved.

We modify the Hessian evaluation function parameter list to include a vector of n elements which specify to the user function which rows of the $n \times n$ Hessian matrix it should evaluate. This allows the Newton algorithm to utilise all the available processors in parallel to calculate different rows of the matrix. The parameter list could be further extended to allow specification of whether the function should calculate rows or columns or even blocks of the matrix. The gradient evaluation function parameter list is similarly extended to specify to the user function which elements of the gradient vector it should calculate.

The user's functions remain largely unaltered except that they should perform checks to ensure that they only evaluate those elements specified. The user-provided function to evaluate F is identical to that required by current sequential algorithms.

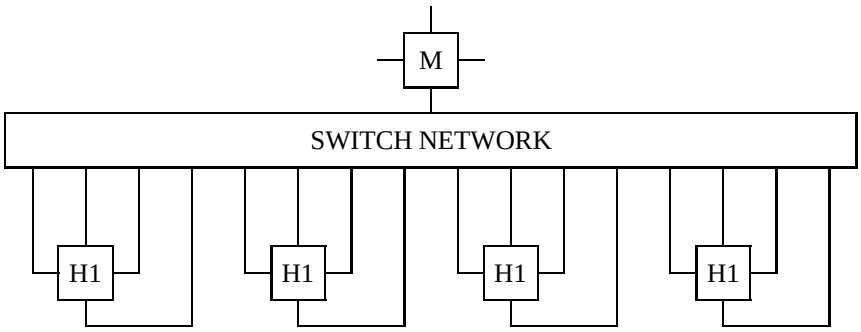

Figure 2: PUMA Transputer architecture

4 Modelling the Algorithm

To develop cost models for the algorithm we need to identify the parameters that characterise the performance of the architectures. For local memory MIMD machines these parameters represent the computation and communication costs. The speed of computation for an individual T8 Transputer will be modelled by a parameter T_{f_1}. This parameter is the time taken to perform a single REAL32 floating point operation. There are several ways in which an actual value for this parameter can be obtained, however we are not interested here in absolute costs but rather in a comparison between the architectures. Computation on the H1 Transputer will similarly be modelled by the parameter T_{f_2}.

Communications on the current generation of Tranputers can be characterised by a single parameter T_{c_1} representing the cost of sending a REAL32 across a link. It has been shown [6, 2] that this single parameter models communication costs well without the addition of a communication startup parameter.

The architecture proposed within the PUMA project supports virtual channels and message through-routing. Our model only takes account of through-routing performed by switch chips: all communications between Transputers are assumed to be through-routed on intermediate switch chips. We allow a constant latency time T_{s_2} for the total bit delay incurred by the switch chips on the message path, and a communication cost T_{c_2} to deliver a single REAL32 across the switch network. The true communication cost may be non-deterministic, as it is dependent upon the congestion in the switch network, but allowing for this in the models is not practicable. Also T_{s_2} will depend upon the number of switch chips on the path between source and destination Transputer. This in turn depends upon the configuration of the switch network and the position of the communicating Transputers on the switch network and the random intermediate switch destination. These factors are not known and so we assume for simplicity that all communications have a latency cost characterised by T_{s_2}. For generality, we assume that each processor has r links (see Figure 2).

We assume further that for both architectures communication on all links and computation can take place simultaneously with no reduction in the performance of individual operations. Appendix A gives cost models and their derivations for some of the common communication primitives based on the preceding assumptions.

5 Gradient and Hessian Evaluation

To model the cost of the user-provided functions to evaluate F, g and G we introduce three parameters T_F, T_g and T_G. T_F is the cost of a single function evaluation. T_g and T_G are the costs for evaluating the most expensive single element of g and G respectively. These parameters all specify a cost in terms of the number of floating point operations required *ie.* a cost given as kT_G means a real time cost of k$T_G T_f$.

The gradient vector and Hessian matrix calculations require no communication and so will have the same cost expression for both architectures. Each of the p processors evaluates n/p rows of G and n/p

elements of g. This has a cost of:

$$\frac{n^2}{p}T_G + \frac{n}{p}T_g \tag{3}$$

6 Solving the Linear System

6.1 Message routing architecture

6.1.1 Forwards elimination

For the forwards elimination phase the following steps are performed with $k = 1, \ldots, n-1$:

- Find best pivot element on each process. Averaged over the $(n-1)$ steps $n/2p$ elements are compared at a cost of:

$$\frac{n}{2p}T_{f_2}$$

- Collect.max operation performed on individual processors' best pivots with k-th row holder as destination (two items are communicated, the pivot holder and the pivot value, but comparisons are made

on the pivot values only):

$$\lceil \log_b((b-1)p+1) \rceil \, (T_{s_2} + 2T_{c_2} + (r-1)T_{f_2})$$

(b is the branching factor of the communications tree, which for processors with r links is given by $r-1$)

- Broadcast of best pivot holder from k-th row holder:

$$\lceil \log_b((b-1)p+1) \rceil \, (T_{s_2} + T_{c_2})$$

- Broadcast $(n-k+1)$ elements of best pivot row and associated right-hand side (RHS) element by its holder:

$$\lceil \log_b((b-1)p+1) \rceil \, (T_{s_2} + (n-k+2)T_{c_2})$$

- Assuming a row swap is required at every step, send $(n-k+1)$ elements of k-th row and associated RHS element to pivot row holder:

$$T_{s_2} + (n-k+2)T_{c_2}$$

- Pivoting of $(n-k)$ rows and RHS elements:

$$\frac{(n-k)(2n-2k+3)}{p} T_{f_2}$$

The total cost for each step is:

$$\left[\frac{2n^2}{p} - \frac{4nk}{p} + \frac{7n}{2p} - \frac{3k}{p} + \frac{2k^2}{p} + (r-1)\lceil \log_b((b-1)p+1) \rceil\right] T_{f_2} +$$
$$\left[3\lceil \log_b((b-1)p+1) \rceil + 1\right] T_{s_2} +$$
$$\left[(n-k+5)\lceil \log_b((b-1)p+1) \rceil + (n-k+2)\right] T_{c_2}$$

Summing over $(n-1)$ steps gives total elimination cost of:

$$\left[\frac{n(2n+5)}{3p} + (r-1)\lceil \log_b((b-1)p+1) \rceil\right](n-1)T_{f_2} +$$
$$(3\lceil \log_b((b-1)p+1) \rceil + 1)(n-1)T_{s_2} +$$
$$\left[\frac{(n+4)}{2} + \frac{(n+10)}{2}\lceil \log_b((b-1)p+1) \rceil\right](n-1)T_{c_2} \quad (4)$$

6.1.2 Backwards substitution

For the backwards substitution phase the following steps are performed with $k = n, \ldots, 1$:

- Holder of row k evaluates element s_k with cost:

$$2T_{f_2}$$

- Broadcast element:

$$\lceil \log_b((b-1)p+1) \rceil \, (T_{s_2} + T_{c_2})$$

- Update subtotals on processors:

$$\frac{2(k-1)}{p} T_{f_2}$$

The total cost for each step is:

$$\left[\frac{2(k-1)}{p}+2\right]T_{f_2}+\lceil\log_b((b-1)p+1)\rceil\,(T_{s_2}+T_{c_2})$$

Summing over n steps gives total backwards substitution cost of:

$$\left[\frac{n(n-1)}{p}+2n\right]T_{f_2}+n\,\lceil\log_b((b-1)p+1)\rceil\,(T_{s_2}+T_{c_2}) \tag{5}$$

6.2 Static configuration architecture

6.2.1 Forwards elimination

For the forwards elimination phase the following steps are performed with $k=1,\ldots,n-1$:

- Find best pivot element on each process. Averaged over the $(n-1)$ steps $n/2p$ elements are compared at a cost of:
$$\frac{n}{2p}T_{f_1}$$
- Collect.max operation performed on individual processors' best pivots with k-th row holder as destination (two items are communicated, the pivot holder and the pivot value, but comparisons are made on the pivot values only):
$$\frac{3(\sqrt{p}-1)}{2}(2T_{c_1}+2T_{f_1})$$
- Broadcast of best pivot holder from k-th row holder:
$$\frac{3(\sqrt{p}-1)}{2}T_{c_1}$$
- Broadcast $(n-k+1)$ elements of best pivot row and associated RHS element by its holder:
$$\frac{3(n-k+2)(\sqrt{p}-1)}{2}T_{c_1}$$
- Assuming a row swap is required at every step, send $(n-k+1)$ elements of k-th row and associated RHS element to pivot row holder. Average distance between source and destination processes is 3 links:
$$3(n-k+2)T_{c_1}$$
- Pivoting of $(n-k)$ rows and RHS elements:
$$\frac{(n-k)(2n-2k+3)}{p}T_{f_1}$$

The total cost for each step is:

$$\left[\frac{2n^2}{p}-\frac{4nk}{p}+\frac{7n}{2p}-\frac{3k}{p}+\frac{2k^2}{p}+\frac{3(\sqrt{p}-1)}{2}\right]T_{f_1}+\left[\frac{3\sqrt{p}}{2}(n-k+5)+\frac{3}{2}(n-k-1)\right]T_{c_1}$$

Summing over $(n-1)$ steps gives total elimination cost of:

$$\left[\frac{n(2n+5)}{3p}+\frac{3(\sqrt{p}+1)}{2}\right](n-1)T_{f_1}+\left[\frac{3\sqrt{p}}{4}(n+10)+\frac{3(n-2)}{4}\right](n-1)T_{c_1} \tag{6}$$

6.2.2 Backwards substitution

For the backwards substitution phase the following steps are performed with $k = n, \ldots, 1$:

- Holder of row k evaluates element s_k with cost:

$$2T_{f_1}$$

- Broadcast element:

$$\frac{3(\sqrt{p}-1)}{2}T_{c_1}$$

- Update subtotals on processors:

$$\frac{2(k-1)}{p}T_{f_1}$$

The total cost for each step is:

$$\left[\frac{2(k-1)}{p}+2\right]T_{f_1}+\frac{3}{2}(\sqrt{p}-1)T_{c_1}$$

Summing over n steps gives total backwards substitution cost of:

$$\left[\frac{n(n-1)}{p}+2n\right]T_{f_1}+\frac{3n}{2}(\sqrt{p}-1)T_{c_1} \tag{7}$$

7 Line search

Before the sequential line search can be performed, the distributed search direction must be collected from the processes at one destination process. Also after the line search the new minimum point x_{k+1} must be broadcast to all the processes for the start of the next iteration. On the through-routing architecture these operations have a combined cost of:

$$\left\lceil\frac{p-1}{r}\right\rceil(T_{s_2}+(n/p)T_{c_2})+\lceil\log_b((b-1)p+1)\rceil\,(T_{s_2}+nT_{c_2})$$

For the current architecture the cost is:

$$\frac{(p-1)n}{3p}T_{c_1}+\frac{3n(\sqrt{p}-1)}{2}T_{c_1}$$

We will assume that the line search fits a third-order polynomial to the function given the function value and gradient at two distinct points. The number of steps required in the line search is problem dependent, but in practice reasonable performance is attained with, on average, 1.5 steps. This gives an approximate cost for the line search of:

$$3(T_F+T_g)$$

8 Example Problem

To give an idea of the relative performance of the two architectures, the cost of a single iteration of each algorithm when applied to a simple test problem is shown. There are many test functions available, but one of the simplest functions of size n (n even) is an adaptation of Rosenbrock's extended function [4]:

$$F(x)=\sum_{i=1}^{n}f_i^2(x)\quad\text{where}\quad f_i(x)=\begin{cases}10(x_{i+1}-x_i^2) & i\text{ odd}\\ 1-x_{i-1} & i\text{ even}\end{cases}$$

This function has a Jacobian gradient vector g given by:

$$g_i(x) = \begin{cases} -400x_i(x_{i+1} - x_i^2) - (1 - x_i) & i \text{ odd} \\ 200(x_i - x_{i-1}^2) & i \text{ even} \end{cases}$$

and a Hessian matrix G:

$$G_{ij}(x) = \begin{cases} 1200x_i^2 - 400x_{i+1} + 1 & i \text{ odd}, j = i \\ 200 & i \text{ even}, j = i \\ -400x_i & i \text{ odd}, j = i + 1; i \text{ even}, j = i - 1 \\ 0 & \text{otherwise} \end{cases}$$

For this function the cost to evaluate the function F is $T_F = (4n - 1)T_f$. The most expensive elements of g to calculate are the odd elements, with a cost $T_g = 6T_f$, and the most expensive elements of G are the diagonal elements on odd rows which have a cost $T_G = 5T_f$.

For the through-routing architecture we assume that the H1 Transputer has 4 links, *ie.* $r = 4$, $b = 3$.

If all these values are substituted into the relevent expressions of Sections 5 to 7 we obtain total cost expressions for each iteration of the algorithms. For the through-routing architecture the cost is:

$$\begin{aligned} &\left[\frac{n}{3p}(2n^2 + 21n + 10) + 4(n-1)\left\lceil \log_3(2p+1)\right\rceil + 14n + 15\right]T_{f_2} + \\ &\left[2(2n-1)\left\lceil \log_3(2p+1)\right\rceil + n + \left\lceil \frac{p-1}{4}\right\rceil - 1\right]T_{s_2} + \\ &\left[\frac{1}{2}(n^2 + 13n - 10)\left\lceil \log_3(2p+1)\right\rceil + \frac{1}{2}(n-1)(n+4) + \frac{n}{p}\left\lceil \frac{p-1}{4}\right\rceil\right]T_{c_2} \end{aligned} \tag{8}$$

and for the current architecture:

$$\begin{aligned} &\left[\frac{n}{3p}(2n^2 + 21n + 10) + \frac{3\sqrt{p}}{2}(n-1) + \frac{31n}{2} + \frac{27}{2}\right]T_{f_1} + \\ &\left[\frac{3}{4}(n^2 - 5n + 2) + \frac{3\sqrt{p}}{4}(n^2 + 11n - 10)\right]T_{c_1} \end{aligned} \tag{9}$$

These expressions are only slightly simpler than the general expressions, but give an impression of the variation of cost with problem size and number of processors.

9 Comparisons

In comparing the algorithms we assume that the computation cost T_f and communication cost T_c are identical for both architectures. The conclusions drawn are only based on order of magnitude comparisons and so are directly applicable to the actual Transputers resulting from the PUMA project provided that these parameters are not varied by more than, say, an order of magnitude.

As is expected the computation costs of the two algorithms are almost identical. The variation in the number of comparison operations performed during communication is only $O(n)$ and is largely insignificant compared with the $O(n^3)$ term for the pivoting operation. In the communication the dominant term for large n and p is $O(n^2 \log_3(2p))$ for the through-routing architecture and $O(n^2\sqrt{p})$ for the current architecture. The $\log$ function grows much more slowly than square root and so for large p the through-routing architecture will have a smaller communication cost. The new parameter to model the startup time for communications on the through-routing architecture has a term of $O(n \log_3(2p))$. Provided that the cost T_{s_2} is not orders of magnitude greater than T_{f_2} and T_{c_2} then this term will be comparatively small for large n and p and so the startup time will not affect the overall cost of the PUMA algorithm significantly.

10 Conclusions

The comparison shows that for this numerical algorithm the PUMA architecture will perform better than the current architecture for large problems, and especially for large arrays, even without taking into account the expected improvement in T_{f_2} and T_{c_2} over T_{f_1} and T_{c_1}. Also revealed is the need for a comprehensive library of efficient communications primitives for both architectures to improve performance, reduce development time for codes and improve portability.

The accuracy with which the performance of the through-routing architecture is modelled depends crucially on factors of the interconnection of switch chips and volume of traffic on the network which affects startup costs. More details of the proposed architecture are required along with network communication performance figures, especially latency figures.

Further work should exploit the symmetry of the Hessian matrix in both the evaluation of the matrix and in the solution of the linear system. The latter will involve replacing the Gaussian elimination algorithm by a Choleski factorisation of the matrix. This factorisation will need to handle indefinite matrices. The algorithms modelled should also be implemented as hardware becomes available to establish the validity of the cost models.

A Communication Primitives

Software developed for local memory MIMD architectures will include code to perform inter-process communications. To simplify development of software, a library of communications primitives should be provided. This would ensure that efficient use is made of the underlying hardware and improve software portability to other MIMD architectures.

This implementation of Newton's algorithm requires three distinct communication operations: broadcast, collect and collect.max. Definitions and cost models for these three operations are given in the following sections.

The communications algorithm used, and hence the cost model, depends on the connectivity of the Transputers. In the design of the Supernode II Library [1] the master process only has a single link connection to the array of slave processes. Hence a different algorithm and cost model are sometimes required when the master process is involved in the communication. Cost models for both cases (with and without master process) are developed where the implementation of Newton's algorithm may involve the master process.

The modelling parameters used are defined in Section 4.

A.1 Broadcast

Definition: One source process sends the same block of data to all other processes.

A.1.1 Message routing architecture

The best process configuration for a broadcast is a b-tree of p processors rooted at the source process (b is the branching factor of the tree which for processors with r links is given by $r-1$). This provides the shortest link path between source and destination processes. At step k in the communication algorithm processes at depth k in the b-tree would output the block of data on all their branches. The total number of steps required to achieve the broadcast is thus the depth of the b-tree, which is given by $\lceil \log_b((b-1)p+1) \rceil$. The cost of the broadcast for a block of size n is:

$$\lceil \log_b((b-1)p+1) \rceil \, (T_{s_2} + nT_{c_2}) \tag{10}$$

This cost can be reduced considerably for large arrays of processors by utilising packet communications (see [6] for more details). In this each block is partitioned into a number of packets of size s and the packets are output from the source sequentially. Processors along the path will input and output packets in parallel.

The destination process inputs the packets and assembles them together again into one block. The cost for this packet broadcast on a b-tree is given by:

$$[n/s + (\lceil \log_b((b-1)p+1) \rceil - 1)]\, T_{s_2} + [n + s\,(\lceil \log_b((b-1)p+1) \rceil - 1)]\, T_{c_2}$$

In general, when algorithms communicate blocks across intermediate processes better performance can be achieved by exchanging block communications for packet communications primitives. Cost expressions for these packet-based algorithms are quite involved and so to simplify the models we have assumed in this paper that only block communications are used.

A.1.2 Static configuration architecture

In the first step in the broadcast algorithm, the source process outputs the data concurrently on all 4 links. In the second and subsequent steps the processes that have just received the data on a link output it on their other links. There will be overheads associated with preventing multiple processes sending the data to one process, but we will not take account of this as the cost would depend upon the implementation.

For a square grid of p processors the maximum number of steps required (occurring when the source processor is at a corner) is $2(\sqrt{p}-1)$. The minimum number of steps possible (occurring when the source processor is centrally located) is $\sqrt{p}-1$. We will assume an average of $3(\sqrt{p}-1)/2$ steps for the algorithm. To broadcast a block of n REAL32 elements would therefore have a cost:

$$\frac{3n(\sqrt{p}-1)}{2} T_{c_1} \tag{11}$$

A.2 Collect

Definition: One destination process receives a different block of data from each other process.

A.2.1 Message routing architecture

For the PUMA architecture the limiting factor in the cost of the algorithm is the number of links into the destination process. The algorithm has $\lceil (p-1)/r \rceil$ steps in each of which the destination process receives directly the data blocks from r different source processes in parallel. The cost for blocks of size n is:

$$\left\lceil \frac{p-1}{r} \right\rceil (T_{s_2} + nT_{c_2}) \tag{12}$$

If the master process is the destination for a collect operation then its single input link is the bottleneck in the network. The cost is then increased to $p(T_{s_2} + nT_{c_2})$.

Distribute, the reverse operation where one source process sends a different block of data to every other process, has an identical cost model.

A.2.2 Static configuration architecture

At each step of the algorithm the destination process inputs a different block of data on each link, beginning with blocks from the nearest processes in the grid. The source processes output their blocks towards the destination and then receive and pass on blocks from more distant processes. This routing of blocks by intermediate processes requires quite complex coding. The number of links into the destination process controls the cost of the algorithm; this can be between 2 and 4 depending on the position of the destination process in the grid. Assuming that, on average, 3 links will be available on the destination process the algorithmic cost for blocks of size n is:

$$\frac{(p-1)}{3} nT_{c_1} \tag{13}$$

Again, if the master process is the destination then the cost is affected in the same manner as for the PUMA architecture with a resulting cost of pnT_{c_1}.

A.3 Collect.max

Definition: Each source process sends a block consisting of a single numeric valued sort field for comparison and an associated data field. The destination process receives the block whose sort field held the maximum value.

A.3.1 Message routing architecture

The cost of this algorithm is similar to the broadcast cost on this architecture. At each step a block is sent from a process and at each intermediate process the maximum valued block received is sent on. The only additional cost over the broadcast is the cost of the comparison operations performed by each process at each step. At each step r blocks have been received by intermediate processes in the tree and are compared. Let the cost of a single REAL32 comparison be T_{f_2} then the comparison cost is $(r-1)T_{f_2}$. The total cost for a block of size n is:

$$\lceil \log_b((b-1)p+1) \rceil \, (T_{s_2} + nT_{c_2} + (r-1)T_{f_2}) \tag{14}$$

A.3.2 Static configuration architecture

The cost model for this operation is again similar to that for the broadcast operation. At each step up to 3 blocks may be received by an intermediate process and require comparing to find the maximum value. The comparison cost is thus at most $2T_{f_1}$ for each step. The algorithm cost for a block of size n is given by:

$$\frac{3(\sqrt{p}-1)}{2}(nT_{c_1} + 2T_{f_1}) \tag{15}$$

References

[1] L. M. Delves and N. G. Brown. The design of the Supernode numerical library. Technical report, Centre for Mathematical Software Research, University of Liverpool, March 1989.

[2] Gabriel N Howard. Solving simultaneous linear equations on transputer arrays. Working paper, Centre for Mathematical Software Research, University of Liverpool, 1988.

[3] INMOS et al. Parallel universal message-passing architectures. Technical annex, ESPRIT Project, July 1989.

[4] Jorge J. Moré, Burton S. Garbow, and Kenneth E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7(1):17–41, March 1981.

[5] W. Murray. *Numerical Methods for Unconstrained Optimization*, page 59. Academic Press, London, first edition, 1972.

[6] Tim Oliver. Calculating eigenvectors on transputer arrays. Working paper, Centre for Mathematical Software Research, University of Liverpool, February 1988.