# PARALLEL SIMULATION OF NONLINEAR NETWORKS USING DIAKOPTICS

**A. Hosseinzaman and A. Bargiela**

Department of Computing
Nottingham Polytechnic
Burton Street, Nottingham NG1 4BU, U.K.

## SUMMARY

This paper is concerned with the evaluation of a new algorithm for simulation of large scale nonlinear network systems as typified by water distribution networks. The algorithm, described in detail in [2], is a development and generalisation of Kron's ideas on tearing linear systems [1] and it combines Newton-Raphson iterative process with diakoptical calculation of state increments at each iteration. While the idea of improving computational efficiency by means of system partitioning is not new, the algorithm for distributed simulation of nonlinear networks and its implementation on a transputer array is novel and is seen as likely to set a trend for modular upgrades of decision support systems. Computational efficiency of the distributed simulator has been evaluated on two realistic networks and the results have been extrapolated to large scale systems.

## 1. INTRODUCTION

The success of computer simulations of water distribution networks during the last decade [3-6], has prompted the industry to consider the development of on-line decision support systems which would integrate the existing telemetry systems and the simulation software. Unfortunately, early attempts to accomplish this task proved to be unsuccessful, largely due to the rapid increase of computational requirements of the simulation algorithms with the increase of the physical network sizes. Even taking full advantage of the sparsity exploiting techniques, the numerical complexity of the nonlinear network solving algorithms is quasi-quadratic with respect to network size. Consequently, the use of the present simulation algorithms on realistic networks would either imply the loss of real-time performance of the decision support systems or it would require computing power which could not be economically justified.

A classical solution to the superlinear numerical complexity problem is the partitioning of the system and, as a further improvement, the use of parallel processing hardware. In adopting this conceptual solution our objective was to develop a suitable distributed simulation algorithm and to implement it on a network of transputers which is both cost efficient and offers expandability of the computing system to accommodate the growth of the physical systems.

The algorithm, described in detail in [2], is a development and generalisation of Kron's ideas on tearing linear systems [1] and, in particular, it combines Newton-Raphson iterative process with diakoptical calculation of state increments at each iteration. In his original work, Kron showed that by splitting up a system into a number of parts, solving the problem on each part separately, and combining the subsystem solutions into an overall solution, an exact answer could be obtained, with a

saving both of computational time and storage space over a direct system solution, even on a sequential computer. More importantly however, the form of decomposition resulting from the application of the Kron's method maps directly onto parallel processing hardware giving rise to an approximately n-fold improvement in the computational efficiency.

The application of Kron's method of tearing (diakoptics) to large-scale linear systems has been reported by several researchers working on such diverse problems as electrical systems modelling, structural analysis, neutron scattering etc, [7-11]. A transputer system implementation of the diakoptics-based solution to Laplace equation on a finite difference mesh has been reported by Bowden [11] but no timing results were published. This paper provides a full set of simulator timings obtained for various subsystem sizes and system partitionings.

Our development of Kron's technique was primarily concerned with overcoming the essential requirement of diakoptics for the linearity of the system which is a prerequisite for finding direct analytical solution. The system linearity ensures that the magnitude of corrections to subsystem solutions is not a limiting factor in the process. By contrast, the nonlinear system, which we analyse, needs to be solved subject to a constraint that the corrections to partial solutions have restricted magnitude so that the local mathematical models of subsystems are not invalidated in the process. The nonlinear system is solved in a sequence of Newton_Raphson iterations with each iterative correction calculated as diakoptical coordinated solution to linearised subsystems.

## 2. THEORETICAL BACKGROUND

The numerical simulation of the system represented on Fig.1 involves the solution of a set of nonlinear mass balance equations for each node in the system. With the flows in network branches $f_{ij}$ represented as

$$f_{ij} = 0.54(x_i - x_j)^{0.54} R_{ij} \tag{1}$$

where $x_i$ , $x_j$ are pressures in corresponding end-nodes and $R_{ij}$ is a hydraulic conductivity of the link, the mass balance equations are expressed as follows

$$\sum_{j \varepsilon \Omega_i} f_{ij}(x_i , x_j) = Z_i \tag{2}$$

$\Omega_i$ is a set of nodes adjacent to node i, and $Z_i$ is a consumption at node i. Equations (2) written for each node in the system can be presented in a more compact form as

$$g(x) = Z \tag{3}$$

where $g()$ is a vector of functions relating inflows/outflows in each node to nodal pressures $x$ and $Z$ represents a vector of consumptions.

The solution to (3) is obtained by linearisation and the iterative improvement of the system state vector $x$

$$x^{k+1} = x^k + \Delta x \tag{4}$$

$$J.\Delta x = (Z - g(x^k)) \tag{5}$$

where $k$ denotes the iteration step and $J$ is a Jacobian matrix, representing sensitivity of mass balance equations to the change in state $x$, calculated for the current estimate of $x$, $x^k$.

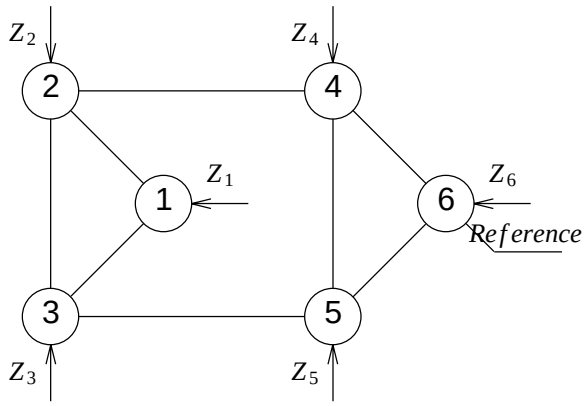


*Figure* 1. *System before partitioning.*

The main computational effort involved in solving (4) and (5) is the inversion of matrix $J$. Even, taking into account the sparse structure of the Jacobian and using suitable sparse matrix factorisation techniques, the computation of $\Delta x$ is quadratically dependent on the size of the Jacobian matrix. It follows, that by partitioning $J$ into blocks, as illustrated in Fig 2. solving (5) "block-at-a-time" and then coordinating partial solutions, computational savings can be achieved.

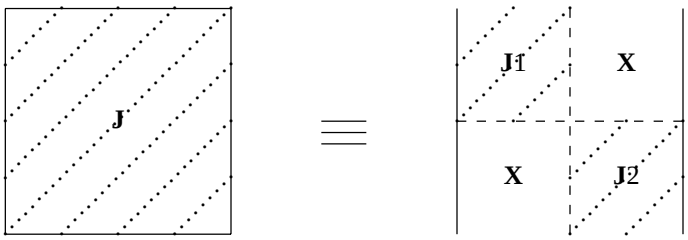


*Figure* 2. *Block structure of the Jacobian matrix.*

However while the "partitioned Jacobian" approach works well when implemented on a single CPU computer it is not best suited for a truly parallel processing architecture as it tends to overburden the coordinating task with additional calculations and sending large amounts of data associated with each Jacobian block. Consequently the parallel processing gains are easily wasted on coordination and communication overheads.

Our approach, proposed in [2], relies on partitioning the network before its linearisation. Figure 3 illustrates the network of Figure 1 partitioned into two subnetworks which are made to behave exactly as the original network by virtue of inclusion of compensating flows $z_2', z_4', z_3'$ and $z_5'$. These flows are calculated separately from the equations describing the removed branches of the network.

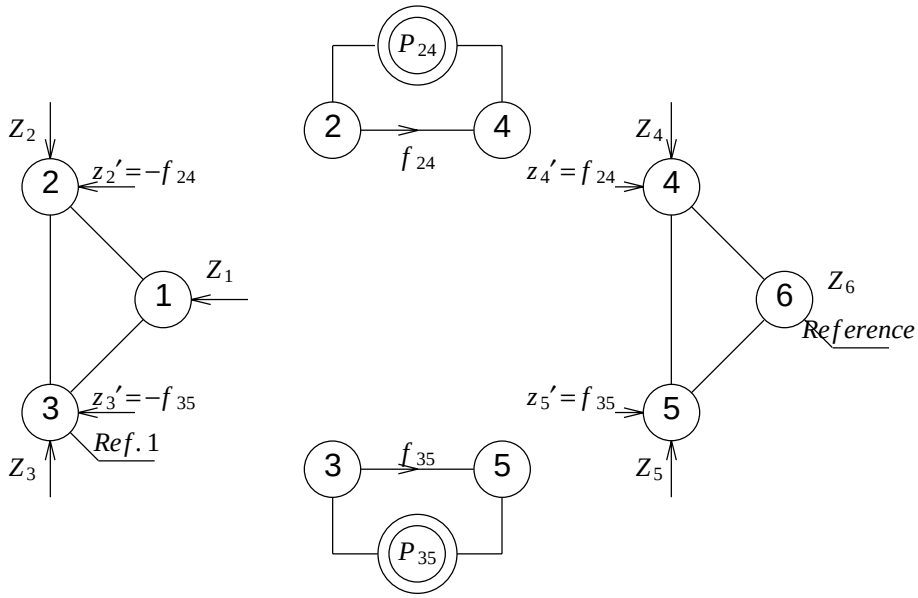


*Figure* 3. *System after partitioning into* 2 *subsystems together with compensating flows.*

The relevant mathematical model describing the network partitioned in such a way is derived from the following conditions:

i) mass balance is preserved in each node,

ii) pressure balance is preserved in the networks formed by the removed branches,

iii) mass balance of consumptions/supplies in each subnetwork and the inflow/outflow through torn branches is preserved.

The above are identical to the conditions pertinent to linear systems [1], but here they have been applied to increments of appropriate variables rather than to their absolute values. These increments have restricted magnitude so that the validity of local mathematical models of subsystems is preserved at all times.

The model of the decomposed linearised system can be expressed as:

$$\begin{bmatrix} J' & -C_{\alpha\psi} \\ C_{\psi\alpha} & K \end{bmatrix} \begin{bmatrix} \Delta x' \\ \Delta y \end{bmatrix} = \begin{bmatrix} Z_0' - g(x_0) \\ Y \end{bmatrix} \tag{6}$$

where $J'$ represents a block diagonal matrix of Jacobians calculated for individual subsystems $J' = \text{diag}\,[J_1', ..., J_n']$, $n$ is a number of subsystems, $C_{\alpha\psi} = C_{\psi\alpha}^T$ is a connectivity matrix between subnetworks and the removed branches and $K$ is a coordinating matrix. $\Delta y$ represents coordinating variables, (increments of flows in removed branches and increments of reference pressures in subsystems) and $Y$ is an augmented vector of zeroes for relevant pressure balances and values of net mass imports/exports for each subnetwork.

The structure of $J'$ facilitates both distributed computation of the component $J_i'$ matrices and the evaluation of the auxiliary solution vector $\Delta x''$

$$\Delta x'' = J'^{-1}(Z_0' - g(x_0)) \tag{7}$$

This auxiliary solution, which represents a collection of solutions to individual subsystems considered in isolation, together with the selected columns of the inverse Jacobian is used in calculating coordinating variables $\Delta y$ and then the coordinated solution to the linearised system $\Delta x'$

$$\Delta y = (K + C_{\psi\alpha} J'^{-1} C_{\alpha\psi})^{-1} (Y - C_{\psi\alpha} . \Delta x'') \tag{8}$$

$$\Delta x' = L(\Delta y) \tag{9}$$

$L$ is a coordinating function.

The above translate onto the following computational algorithm:

Step 1 : Read-in the system description data.
Step 2 : Form subsystem data packets and send them to individual solvers.
Step 3 : Calculate subsystem solutions (Equation 7).
Step 4 : Coordinate partial solutions (Equations 8 and 9).
Step 5 : If the coordinated corrections from Step 4 are less then a given value then STOP otherwise repeat from Step 2.

## 3. PROGRAM STRUCTURE

The structure of the nonlinear diakoptics based, water system simulation program is shown in Figure 4. After initial reading of system data by module 1.1, packets of data describing individual subsystems are being sent by module 1.2 to multiple copies of network solvers 1.3. The results produced by modules 1.3 are transferred to module 1.4 which collates packets of results arriving from individual solvers in an arbitrary order and then sends a complete set of results for coordination to module 1.5. If the coordinated state estimates satisfy the convergence criterion the results are output, otherwise the state estimate is transferred to module 1.2 and the cycle of computations is repeated.

Since the program is intended to run on a distributed multiprocessor system the issues of minimization of communication overheads and coordination of concurrent execution of program modules had to be addressed. Considering a single iteration through the simulation loop, it is apparent that the amount of data transferred between modules 1.4-1.5 and 1.5-1.2 is approximately n-times greater then the corresponding data transfers between modules 1.3-1.4 and 1.2-1.3 (where n denotes the number of active processors executing module 1.3). Consequently, in order to minimise the communication overheads, the data transfers 1.4-1.5 and 1.5-1.2 have been made as efficient as possible by placing modules 1.2, 1.4 and 1.5 (together with 1.1 and 1.6) on the same processor and making use of common memory access for implicit data transfers. Such facility is available if modules 1.2, 1.4 and 1.5 are implemented as *threads* rather than tasks (3L Fortran) [12].

By contrast, communication between modules 1.2-1.3 and 1.3-1.4 involves explicit data transfers since different copies of 1.3 are executing on different processors, in order to maximise the benefits derived from the parallel structure of the algorithm. For these data transfers, communication overheads can be minimised only by minimising the amount of data which is actually sent. Our algorithm achieves considerable reduction in data traffic between modules 1.2 and 1.3 by calculating Jacobian matrices within the module 1.3, and by sending from 1.3 to 1.4 only these columns of the inverse of the Jacobian which correspond to the end-nodes of the torn

branches. The latter, while not implemented in [11], has been recognised as a major improvement to the diacoptics algorithm as it reduces the corresponding data traffic by a factor of 10 to 100 for a typical large-scale system.

In general, the number of processors available for the execution of subsystem solvers 1.3 is not equal to the number of subsystems onto which the system is subdivided, therefore there is a need to coordinate the concurrent execution of sending and receiving threads. While the low-level synchronisation of 1.2-1.3 and 1.3-1.4 is provided from within 3L Fortran procedures, F77_NET_SEND and F77_NET_RECEIVE, it is up to the application program to ensure that no solver 1.3 remains idle while some packets of data still need to be processed.

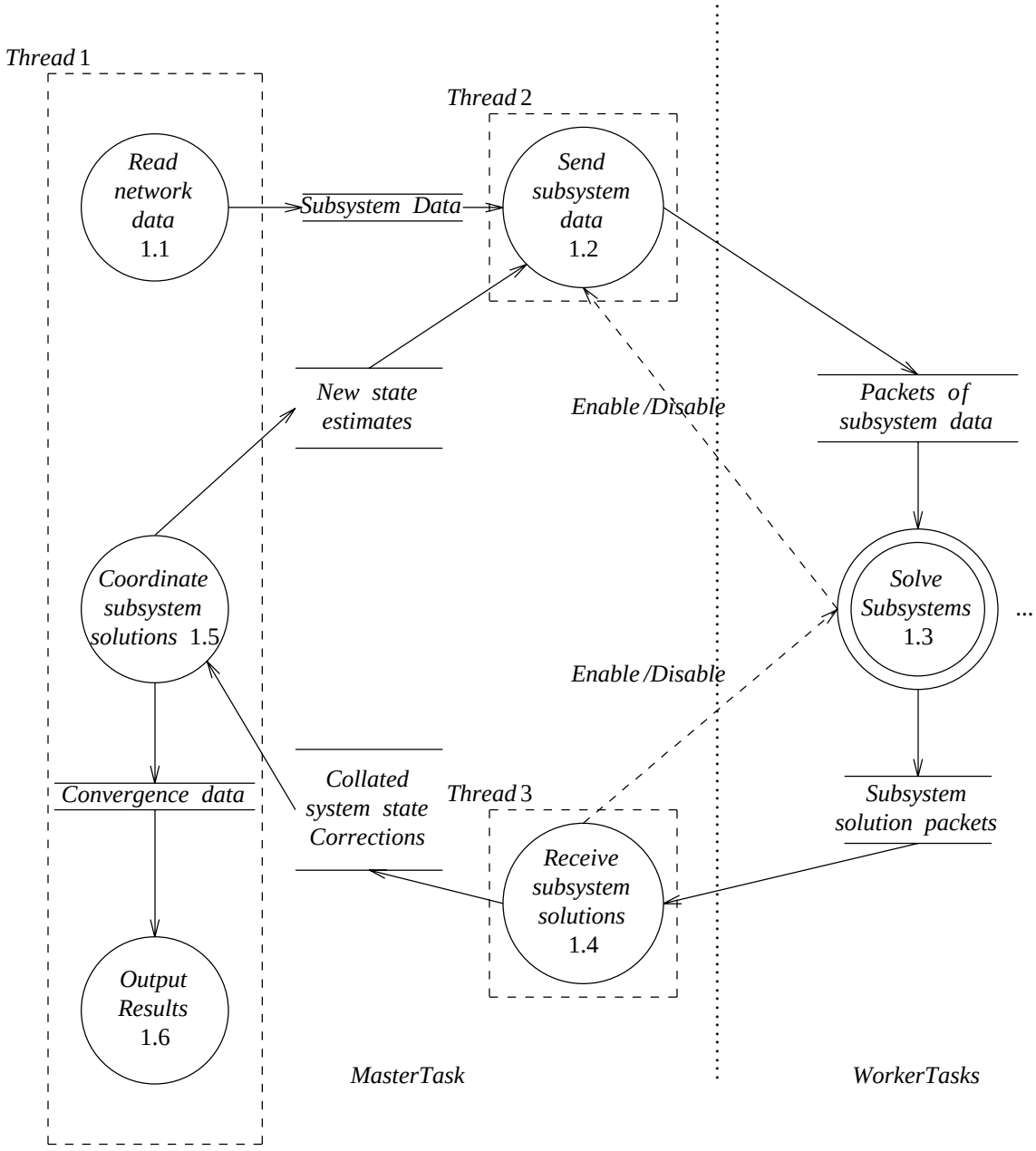


*Figure 4. Water distribution simulation program*

It was originaly envisaged that the maximisation of the use of available processors will be facilitated by the use of the standard 'Flood Configurer' [12]. However, our experiments indicate that in the cases where the number of transputers is approximately equal to the number of subsystems, the Flood Configurer grossly underutilises the system leaving most of the processors idle. Consequently, additional routing code was developed and included within the relevant modules (1.2, 1.3, 1.4).

Initially the solution process starts by sending data packets to all available processors. When a result packet is received by 1.4 the processor which just finished the execution of 1.3 becomes available. In order to maximise the computational efficiency, the module 1.2 needs priority-schedulling so that the free subsystem solver receives its data and starts without delay. This immediate resumption is achieved by designating the thread executing module 1.2 as 'urgent' (higher priority than other threads) and forcing de-scheduling of threads every few milliseconds.

## 4. PROCESSOR STRUCTURE

The algorithm described in the previous section maps well onto a tree structure of processors where the *root* processor executes threads 1, 2 and 3, and the tree branch processors *(workers)* execute the subsystem solvers, 1.3. Given that each transputer provides only four links for interconnections with other transputers, a two-layer tree of transputers imposes a limit of three concurrent subsystem solvers. If a greater number of modules 1.3 is to be executed concurrently, transputers need to be connected as a multi-layer tree where each transputer of the lower layer is connected to up-to three transputers of the higher layer. A three-layer-tree will then accommodate up-to 12 concurrent solvers, a four-layer-tree, up-to 33 solvers, etc.

An alternative pipeline configuration is seen as a 'general purpose' configuration which, in particular, can be used for the execution of our algorithm but it implies the overhead of a greater data traffic through individual transputers. Figure 5 gives the graphical representation of the performance of pipeline and tree configurations.

Let's consider a four-processor system (root and 3 workers W1, W2 and W3) and represent the time required for the transfer of subsystem data as 'D', the subsystem solution time as 'S' and the time required for the transfer of results as 'R'. In the pipeline configuration, the root transputer sends 3 data packets D to worker W1. W1 retains one packet of data and passes on the remaining two packets to W2. Similarily, W2 retains one data packet and passes on the other packet to W3. The total time spent on propagating data packets is 6D. Assuming now that the subsystem solution times S are identical, W3 will be the last to complete its calculations and in the worst case scenario, where there is no concurrent transmission of results, the time required to propagate packets of results to the root transputer will be 6R giving a total time $t_{tot}=6D+S+6R$.

A 2-layer tree configuration avoids entirely the need for the re-transmission of data packets by the workers so the maximum delay incurred in starting the third worker is 3D; a saving of 3D compared with the pipeline configuration. While the subsystem solution times are as before, the transmission of results back to the root transputer does not involve other workers and can be accomplished concurrently with subsystem solutions giving rise to the best time attainable in the above tree configuration of $t_{tot}=3D+S+R$, if D is greater then R, or $t_{tot}=D+S+3R$, if R is greater than D. Consequently the maximum performance advantage of the tree over the pipeline configuration is $3D+5R$ or $5D+3R$ depending on the relative length of D and R.

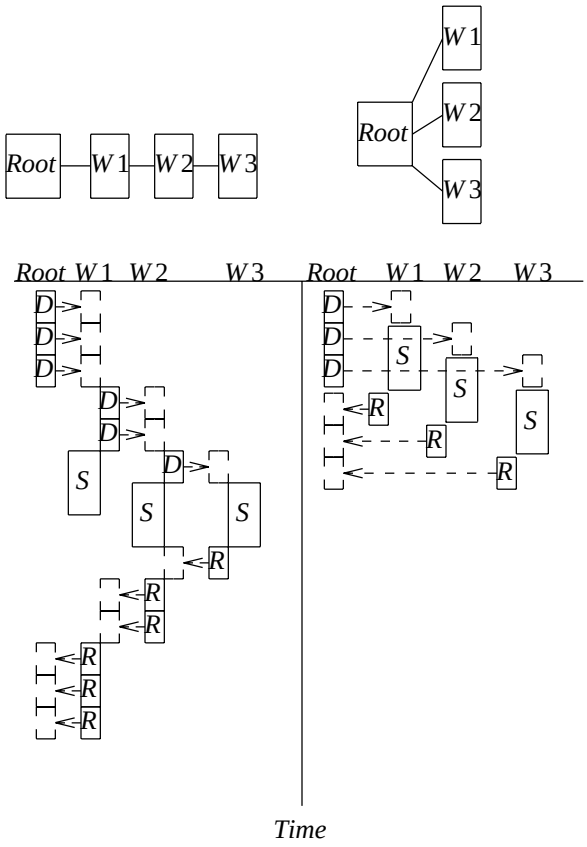


*Figure* 5. *Performance of Tree* and *Pipeline configurations*

In the context of realistic 65 and 130 node networks (the latter illustrated in Figure 6), the relevant timings were performed. Times D and R were found to be of the order of 1-2ms and the subsystem solution times (3 subsystems) were found to be 780 and 1780ms for the two systems respectively. It is not surprising therefore that the expected performance advantage of the tree configuration, of 8-16ms, was seen at best as marginal compared with the subsystem solution times, implying that our distributed simulation algorithm is not sensitive to transputer configuration. These results are corroborated by the findings of other researchers, [11], who reported that the tree configuration of processors executing diakoptics algorithm, does not offer significant advantages over the pipeline configuration. However, unlike in [11], with our implementation of diakoptics we did not find the evidence of a ’data transfer bottleneck’ in the tree configuration, but instead, we could explain all the results with the efficiency of data transfer (small volumes of data and fast links).

## 5. COMPUTATIONAL RESULTS

The performance of the water system simulation program was evaluated on two realistic 65 and 130 node networks (see Figure 6). The algorithm was coded in 3L Parallel Fortran [12] and was run on a reconfigurable 5-transputer system hosted in a ’486-based Unix workstation. The tests were performed for both pipeline and tree configurations of transputers but, as explained in the previous section, the two sets of results were not sufficiently different to merit separate consideration.

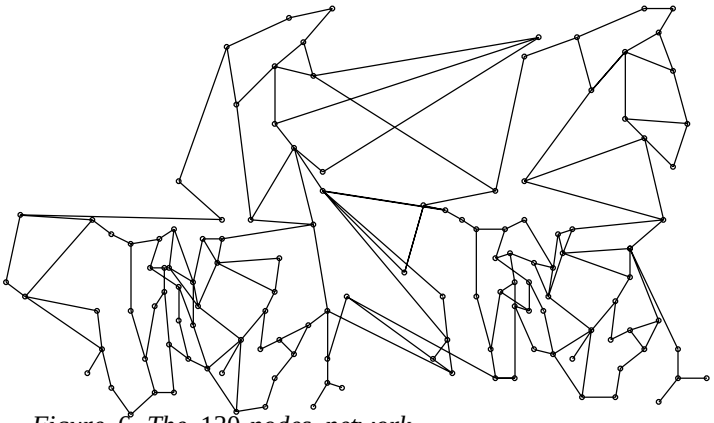

*Figure* 6. *The* 130 *nodes network.*

In order to gain some insight into the computational efficiency of the algorithm itself and to establish basic reference data for the future investigation of the optimal network tearing the following were investigated:

i) coordination time vs. the number of torn branches and

ii) subsystem solution time vs. the subsystem size.

The networks were partitioned into 2, ..., 5 subnetworks in several different ways, varying the number of torn branches. For each partitioning, an attempt was made to have subsystems of approximately the same size so that the allocation of subsystems to processors did not affect the results. In the case of 2, 3 and 4 subsystems the subsystem solver tasks were placed on the 'worker' transputers only, but in the case of 5 subsystems, a subsystem solver task was also placed on the root transputer. It is important to note that the solver tasks had to be individually placed on the selected transputers and the data packages had to be appropriately managed by a separate router software since, as we have found, the standard 'Flood Configurer' [12] tended to send data packets only to some of the 'workers' leaving others idle.

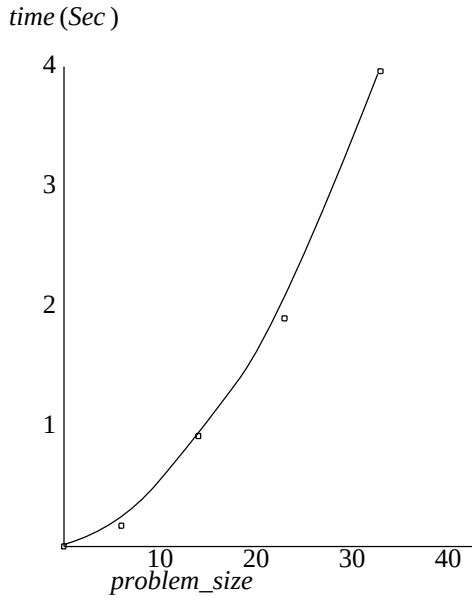


*Figure* 7. *Coordination time /problem_size*

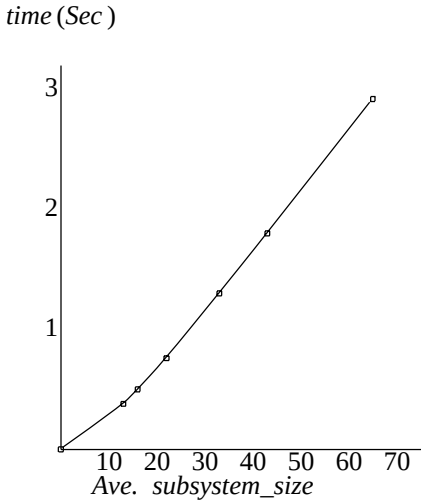


*Figure* 8. *Subsystem Solution Times*

The time required to coordinate subsystem solutions, (Figure 7), was found to depend almost quadraticaly on the coordination problem size, as defined by the combined count of subsystems and the torn branches between them. On the other hand, the subsystem solution times were found to depend quasi-linearily (power 1.25) on the number of nodes in subsystems (Figure 8). These results indicate that while it is advantageous to solve subsystems concurrently one needs to be careful about subdividing the systems into too small subnetworks, since the implied increase of the coordination time may outweigh the concurrent processing gains. Indeed, for a 130-node network subdivided into 2 subsystems, 2.920sec was needed for subsystem solution and only 0.189sec for coordination of partial solutions, giving a total of 3.109sec. The same network, subdivided into 4 subsystems, required 1.300sec for concurrent solution of subsystems and 1.895sec for coordination if the number of torn branches was minimised (19) and as much as 3.975sec if the number of torn branches was high (29). This gives corresponding totals of 3.195sec and 5.275sec. Consequently, it is apparent that the full benefits of the distributed simulation algorithm will be derived from its application to large scale systems.

By way of an example, the simulation of a 1300-node network partitioned into five subsystems with, say, 30 torn branches (the number of torn branches does not depend on the network size but only on the number of subsystems and the average node connectivity) will still require approx. 4secs for coordination of subsystem solutions, and the concurrent execution of five 260-node subsystems is estimated to require 17 secs giving a total of 21 secs. This compares very favourably with the estimated 90secs needed to simulate this system on a single transputer using our algorithm and 130secs needed if the network tearing algorithm is not used. It is worth pointing out that the volume of data transmitted between tasks is linearily proportional to the network size, so the data transfer times will continue to be negligible.

## 6. CONCLUSIONS

This paper discussed a transputer implementation of the nonlinear network simulation algorithm [2]. Computational results confirm that the algorithm is well suited for large-scale systems and the transputer platform provides a good vehicle for its implementation.

It has been found that a special care is needed with the routing of data packets to individual worker-transputers in order to maximise the utilisation of the transputer system.

## ACKNOWLEDGEMENTS

The authors wish to acknowledge an efficient support and help provided by 3L Ltd. in overcoming the limitations of the standard 'Flood Configurer'.

## REFERENCES

1. KRON G., -'Diakoptics: The piecewise solution to large scale systems', Macdonald, 1963

2. BARGIELA A., - Nonlinear network tearing algorithm for transputer system implementation, preprints available from the author

3. COULBECK B., STERLING M., - Optimised control of water distribution systems, IEE Proc., Vol 125, Oct. 1978.

4. STERLING M., BARGIELA A., - Minimum norm state estimation for computer control of water distribution systems, IEE Proc., Part D, Vol 131, March 1984, pp. 57-63.

5. BARGIELA A., HAINSWORTH G., - Pressure and flow uncertainty in water systems', ASCE Journal of Water resources Planning and Management, Vol 115, March 1989, pp. 212-229.

6. JOWITT P., XU C., - Optimal valve control in water distribution networks, ASCE Journal of Water Resources Planning and Management, Vol 116, July 1990, pp. 455-472.

7. CHUA L.O., CHEN L. K., - Diakoptics and generalised hybrid analysis, IEEE TRANS. Circuit and Systems, Vol. CAS_23 pp 694-705, Dec. 1976.

8. ONODERA R., - Diakoptics and Codiakoptics of electrical networks, RAAG MEMOIRS Vol.2, 1958.

9. HAPP H.H., - Diakoptics_ The solution of system problems by tearing, Proc. IEEE, 62, 7,pp 930-940 July 1974.

10. WU. F. - Solution of large_scale networks by tearing, IEEE Trans. Circuit and systems, CAS_23,12,pp.706-713 Dec.1976.

11. BOWDEN K., - Kron's Method of Tearing on Transputer Arrays, The Computer Journal, Vol. 33, No.5, 1990.

12. PARALLEL FORTRAN User Guide, version 2.1.4, 3L Ltd.