FOR REFERENCE ONLY

PKD

QS / MOS

SLC

Ref.

THE PARALLEL AND DISTRIBUTED SIMULATION OF NETWORK SYSTEMS

by

A. HOSSEINZAMAN

A thesis submitted in partial fulfilment of the
requirements of The Nottingham Trent University
for the degree of Doctor of Philosophy.

June 1995

Acknowledgements

I should like to thank my supervisor
Dr Andrzej Bargiela for his constant
help and encouragement throughout the
duration of the project.

ABSTRACT

Successful computer simulation of water distribution system prompted the industry to consider development of on_line decision support systems which would incorporate existing telemetry systems and simulation software. Unfortunately, early attempts to accomplish this undertaking proved to be unsuccessful, largely due to the rapid increase of computational requirements of the simulation algorithms as a result of the increase in the physical network size.

In spite of using sparsity exploiting techniques, the nonlinear network solving algorithm demonstrates quadratic numerical complexity therefore is unable to cater for network growth.

An original approach to the solution of such systems would be to partition the overall problem into smaller units and solve these in isolation(and as a further improvement use parallel processing in solving the derived subsystems). Once the solution of the smaller units are known, they are combined in a coordinating routine to yield the overall solution.

The main objective of this research was therefore, to develop a suitable distributed simulation algorithm and to implement it in a distributed computing system. The research resulted in the development and implementation of a nonlinear diakoptics algorithm. This research has established a framework for the development of a distributed computing system based on the generalization of ADA rendezvous mechanism.

The overall distributed computing scheme was implemented on a tightly-coupled transputer network and on a network of loosely-coupled workstations connected using Ethernet communication link. The computational efficiency of the algorithm is evaluated using two realistic networks and results are extrapolated to large scale systems.

The overall computational efficiency and the amount of storage required in the network tearing method is strongly influenced by the way the system is partitioned. Two graph partitioning techniques were evaluated and their performances in partitioning the given example networks are compared and presented in the form of graphs in which derived partitions are clearly depicted.

DEDICATION

I dedicate this thesis to my parents without whose love and support none of this would have been possible.

Glossary of terms

$A^{(i)}$	Non-Zero elements in column i.
AS	Adjacent set.
A	Energy coefficient matrix in linear theory method.
B	Pipe flow incident matrix in linear theory method.
C	Cost function.
$C_{\alpha\psi}$	Node/cut pipe incident matrix.
CN	Contour number.
ΔC	Cost difference.
D	Diagonal elementry matrix.
ΔE	Energy difference.
F(),f()	Function of.
F'	Partial derivative of function F.
Δf_{ψ}	Rate of change of flow in pipe ψ.
g(x)	Vector of mass balances.
$g_i()$	Mass balance in node i.
H	First derivative of pressure with respect to flow in pipe ψ
h(f)	Functional relationships between the flow and the pressure drop.
IS	Iterating set.
J	Jacobian matrix.
J'	Similar to J but without the interconnecting elements between the subnetworks.
L_k	Lower kth column elementry matrix.
L	Lower triangular matrix.

$L^{(i)}$	Left hand factor ith matrix.
$M_{\psi\delta}$	The cut pipe/subnetwork incident matrix.
M_i	Matrix which differs from Identity matrix in just one off-diagonal element.
P,Q	Permutation matrix.
P_ψ	Quantifies the pressure drop in terms of the difference of pressures in the end-nodes of the pipe and the difference of reference pressures x_δ in the relevant subnetworks.
Q_j	Flow in pipe j.
R_{ij}	Hydraulic resistance of the i-j pipe.
$R^{(i)}$	Right hand factor ith matrix.
T	Temperature.
T_k	Column elementry matrix.
X_i	Pressure at node i.
Δx	Pressure difference.
$\Delta x''$	Uncoordinated subsystem solutions.
U IS(j)	Union of the Iterating set over the range j.
U	Upper triangular matrix.
W(i)	Collection of nodes not adjacent to Z(i) nodes.
Z_δ	Sum of consumption (z_i) in all the nodes of subnetwork δ.
Z(i)	Collections of nodes not adjacent to W(i) nodes.
Z_i	Consumption at node i.
Z	Vector of consumptions Z_i.
Z'	Consumption vector of the derived network due to partitioning.
z'	Consumption vector of the cut-network.
Z_0	Consumption vector derived from the sum of consumption in the cut-networks and the partitioned subnetworks.

Ω_i Set of nodes adjacent to node i.

List of Figures **Page No.**

List of Tables **Page No.**

CONTENT

CHAPTER 1: Introduction to Water distribution systems.

1.1 Aims and Objectives

Successful computer simulation of water distribution system prompted the industry to consider development of on_line decision support systems which would incorporate existing telemetry systems and simulation software. Unfortunately, early attempts to accomplish this undertaking proved to be unsuccessful, largely due to the rapid increase of computational requirements of the simulation algorithms as a result of the increase in the physical network size. To remedy this shortcoming, sparsity exploiting techniques were incorporated in the simulation algorithm. These included "near_optimal pivoting" and "sparsity directed storage"techniques.

Although these techniques are efficient in exploiting the inherent sparsity of the water distribution systems, the inability of nonlinear network solving algorithms (incorporating these techniques) in accommodating for network growth means that alternative solution methods are required.

A standard approach to the solution of such systems is to partition the overall problem into smaller units and solve these in isolation (and as a further improvement use parallel processing in solving the derived subsystems). Once the solution of the smaller units are known, they are combined in a coordinating routine to yield the overall solution. This approach is particularly suitable for systems such as water distribution networks which are constructed from individual groups of highly connected nodes, the individual groups being

only sparsely interconnected. For example, in the case of local water authorities, the city can be divided geographically into several regions (i.e dense industrial or domestic zones), with a few interconnecting links. The derived regions can then be mapped onto the available local processor units in the computer network, resulting in a truly distributed processing scheme. The overall solution is then achieved by the coordinating processor which collates and re_adjusts the subsystem solutions.

The computational efficiency of such a distributed processing scheme is clearly related to the number of processing nodes but is significantly effected by the efficiency of the communication links/amount of transferred data between the computing nodes as well as the efficiency of the coordinating task.

The main objective of this work was, to develop both a distributed simulation algorithm and a suitable distributed computing environment. This research has demonstrated that the concept of "virtual node" **[5]** for modelling the distributed processing nodes of a loosely-coupled system, is a viable approach. The "Virtual Node" approach was selected amongst other methods because: (i) it offers a compiler independent virtual node approach, therefore conventional Ada compilers can be used to develop the distributed application program, (ii) it allows the notion of virtual node type, which is particularly useful in this implementation since it offers dynamic creation of processing nodes to fit a particular partitioning scheme, and (iii) it uses a rendezvous like mechanism for inter-virtual node communication - useful since rendezvous mechanism gives a better reflection of the work-load of the processing nodes in the system.

1.2 Review of Thesis

The early attempts to develop on-line decision support systems for monitoring water distribution systems had only limited success. This was due to the rapid increase of computational requirements of the simulation algorithms as a result of an increase in the physical network size. To overcome these shortcomings sparsity exploiting techniques were incorporated in the simulation. These techniques are discussed in detail in Chapter 2. They include: Sparsity directed matrix inversion routines, Bi-Factorization, Bartels-Golub decomposition, sparsity-directed storage techniques (i.e. Link-Lists), and near-optimal pivoting techniques.

However, further investigations in order to determine the efficiency of these techniques, in solving the derived system matrix revealed that their performance is highly dependent on two factors, firstly is the pivotal strategy used in order to achieve minimum "fill-in" thus reducing storage requirements and achieving speedup, and secondly is the storage technique itself. These techniques would only reduce the computational requirements of the simulation algorithm in the short term, and therefore can not be used as a long term measure for reducing the computational requirements of the algorithm in the face of continual network expansion. Chapter3 addresses this problem and develops a new nonlinear parallel processing algorithm (i.e. nonlinear diakoptics).

The algorithm is initially run on a closely-coupled computing system (PC-based Transputer System configured in "Tree" and "Pipe-line" modes) and its computational efficiency is evaluated using two realistic networks (i.e. 65 node and 130 node networks).

The result of these implementations indicate that the overall computational efficiency and storage requirements of the nonlinear diakoptics algorithm is strongly influenced by the way the system is partitioned. Chapter 4 introduces two graph partitioning techniques in order to achieve optimum partitioning of the system. These techniques belong to two theoretically different categories of algorithms. The first method known as "Greedy" algorithm **[45,89]** has its basis in heuristic derivation and the second method known as "Simulated Annealing" is based on the analysis of combinatorial optimization problems.

A modern water distribution and monitoring system consists of a number of local operational control centres communicating with one another over communication lines. This arrangement of processing power form a loosely-coupled computing system **[6]**. Chapter 5 introduces the distributed computing systems and in particular concentrates on loosely-coupled systems. Furthermore, it identifies those characteristics of a water distribution and monitoring system that are best suited to a loosely-coupled computing system. Characteristics such as the need for periodic expansion of water distribution systems which can be catered for by simply increasing the computing power (i.e. by adding more computers to the network) to deal with network's topological expansion.

Moreover, in order to develop application programs for loosely-coupled systems, the programming language must have certain characteristics. They are: software configurability, inter-process communication mechanism(synchronous or synchronous) and finally the partial failure mechanism. The suitability of each language would be measured on how many of these requirements it can satisfy. The Ada language satisfies most of the

requirements, however, it is generally acknowledged that the language support in the area of distributed systems is lacking **[5]**.

This research establishes a framework for the development of a distributed computing system based on the generalisation of the ADA rendezvous.

Chapter 6 presents the implementation of a distributed computing environment for water system simulation, which serves to validate the general concept of distributed computation using ADA Virtual Nodes. Major functional units of the system are identified and diagrammatically defined - namely the coordination and subsystem solution routines. This Chapter (i.e. Chapter 6) demonstrates that the concept of "Virtual Node" for modelling the distributed processing nodes of a loosely-coupled system is a viable approach.

1.3 Water distribution monitoring systems.

The early water distribution systems employed very basic measurement devices which monitor only few parameters in the system such as levels of water in reservoirs or pump status. The distribution network itself was largely unmonitored due to the difficulty of interpretation of the measurements by the human operator, without the aid of extensive computer_based water system simulations.

However, water distribution systems have come a long way since then so that now, for the current systems under consideration the data procured would not only be monitored, but also evaluated and used to provide predictions, reports and operational control, thus

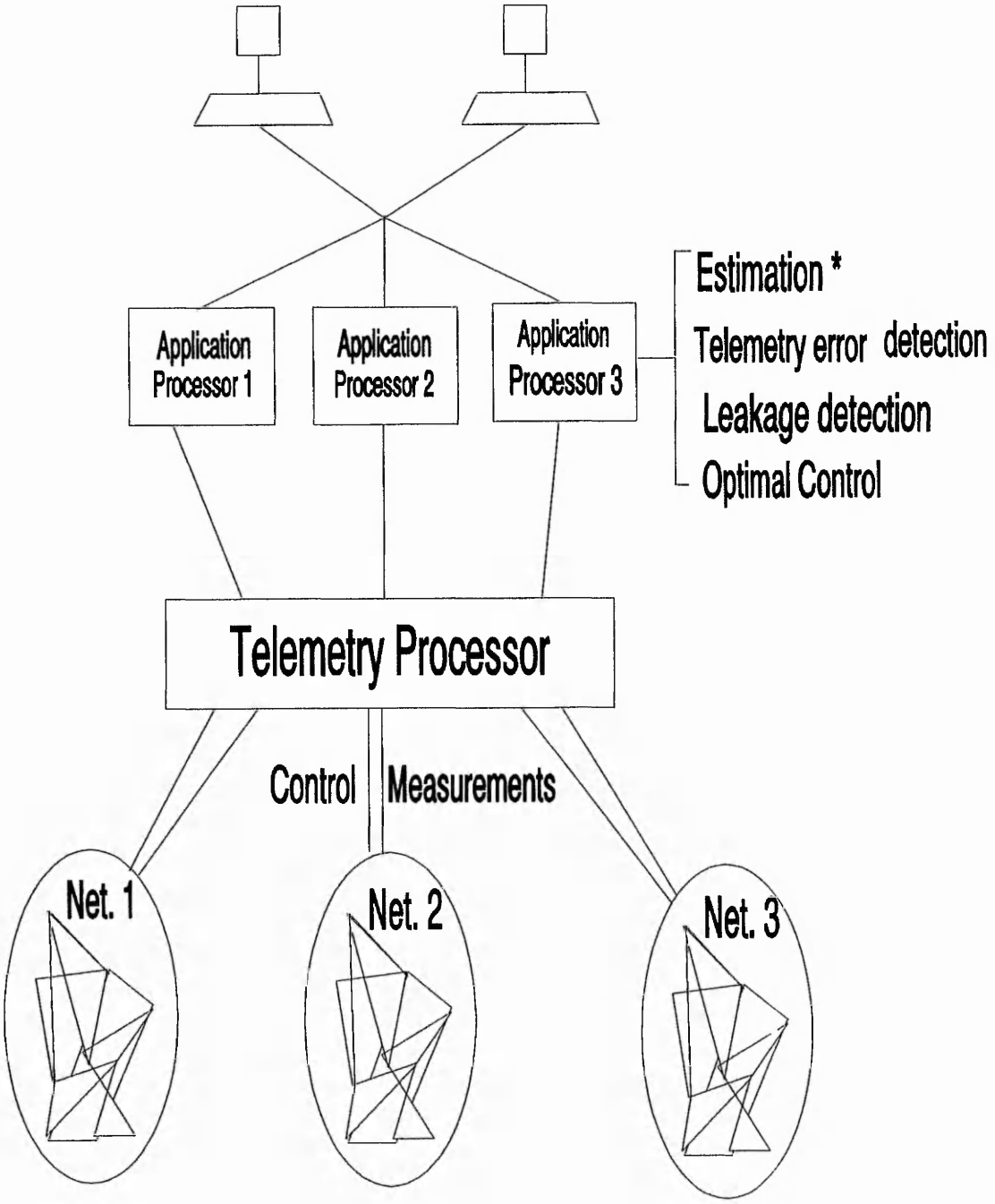

Figure 1: Water distribution monitoring system.

bringing about more effective and near optimum operation of sites, in addition to providing comprehensive information to managers and users. Fig.1 shows various components of a telemetry system. Several separate monitoring outstations exist within the water company, operating independently of each other and covering the different water supply and distribution areas (i.e Net1,Net2,Net3). In addition to these distributed multi_site systems self contained SCADA systems are in operation, usually autonomously, at large water treatment works and reclamation centres. The data is collated by the telemetry processor and distributed amongst the application processors for telemetry error detection, leakage detection and optimal control. However, the pivotal application in the on-line decision support system for the simulation of water distribution system is the state estimation.

The expansion of the monitoring systems is mirrored by the expansion of the mathematical model of a network and consequently may imply additional processing load. In this work the emphasis is on water network simulations only which require minimum measurement set. The minimum measurement set is comprised of the mass balance equations for all the nodes in the network except the reference node, plus one pressure measurement in reference node.

1.4 SCADA system definition.

Amongst several systems that provide a means for "data acquisition" are**[144]**, the traditional master/RTU SCADA systems, Distributed Control Systems, Programmable Logic Controllers, PC-based SCADA systems, Emergency shutdown systems, and Fire

and Gas Control Systems. The term Supervisory Control may not be exactly applicable in all instances but using the definition given below it covers almost all system control categories. A supervisory control function is defined as a higher level control that interfaces with a regulatory controller to provide integrated and/or remote control. Regulatory control is defined as execution of a control algorithm, based on measured input signals, and transmitting an output value to a field control device in order to provide closed loop control of that process.

Since intelligent RTUs, distributed control units and programmable logic control units provide discrete and regulatory control functions they all fall under the term Supervisory Control and Data Acquisition.

1.5 Computer Applications in Water Industry.

The computer applications in water industry are wide spread, in particular computers are used in the areas such as on-line monitoring and control, automated mapping/facilities management (AM/FM), and Geographic Information Systems (GIS). The on-line monitoring and control acquires up-to-date information about the operation of distribution networks and treatment works.

The AM perform project-specific designs or mapping of facilities in graphic environment with limited nongraphic data analysis. FM supports facilities inventory, management and analysis without a sophisticated graphic display capability. GIS supports a wider range of AM/FM applications such as network modelling, incident mapping and polygon overlay

analysis. The overall picture of activities in these areas show a high level of interest in computer applications amongst water utilities. Most utilities require more information about integration among systems and data bases. Key areas of interest are integration of FM systems and data bases with GIS data, interface of GIS with water models, and sharing and transfer of data between GIS or AM/FM systems and CAD. Furthermore, the FM and GIS systems are known**[109]** to have been linked or attempts are made to link them to their Supervisory Control and Data Acquisition (SCADA) systems.

The degree of computerization and interest in new technology within the water industry is high. The benefits of interfaced computer systems seem to be well realized within the industry. Although many utilities are investigating AM/FM and GIS and are interested in new technologies such as electronic document management system (EDMS), most are still in their preliminary stages of implementations.

1.6 Telemetry system operation.

The early water network simulation systems were composed of a number of remote stations whose tasks were to monitor and gather data, and a main computer centre. The data(pressure/flow measurements and occasionally consumption) sent by the remote stations is utilized by the network simulating software installed at the telemetry computer centre, yielding directives for the overall control of the network. The processing power is concentrated at the telemetry computer centre, thus leaving the remote stations with little or no processing power. Information procurement was on a continuous polling of remote stations basis ("time skew"), with monitored data reading being updated every few

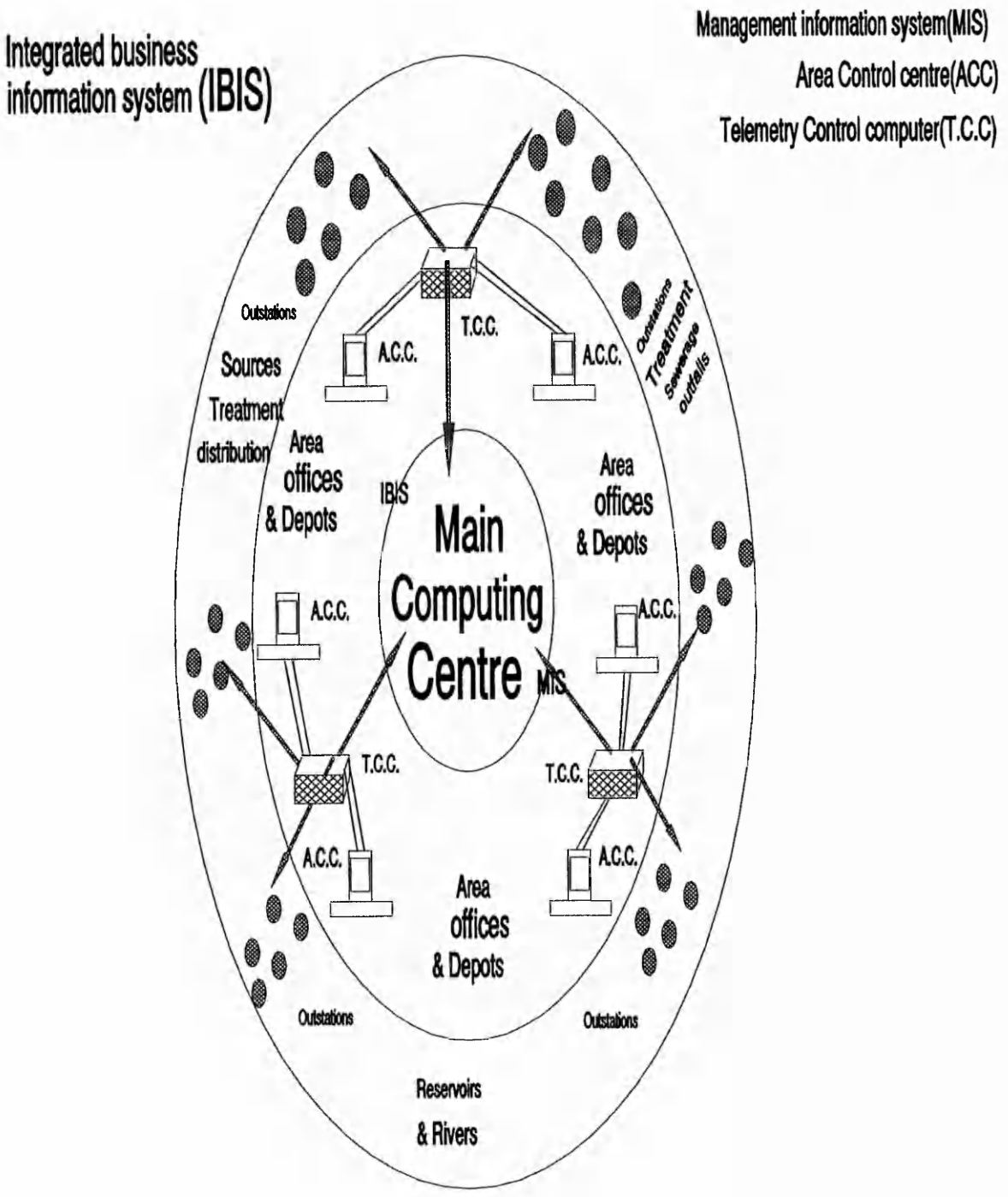

Figure 2: Welsh water's telemetry and information management system.

minutes. However, this method of monitoring as mentioned before was not equipped to deal with network growth and proved to be fatal in the face of failures, since a computer shutdown would mean that backup hardware needed to be operational immediately.

In the new telemetry systems however, the remote stations are equipped with powerful microcomputers or workstations capable of running simulation algorithms locally at the remote site. This way, communication between the remote stations and the central unit is reduced to merely call for raising alarm conditions or selected data transmission (which would be small in volume). Information procured by the main station is displayed in the normal way on computer display terminals operating sophisticated high resolution graphics. Fig.2 shows the Welsh water's telemetry and information management system using the newer telemetry systems. The data which originates from telemetry remote stations is processed by the simulating software installed at the telemetry computer centre (TCCs).

Another system in operation for the control of city of Hull's water distribution system is shown in Fig.3. The system has a pre-set pressure profile in order to prevent pressure rises above that can be tolerated by the 100 year old pipe network. The Decision Support System(DSS) employed in the project, on detecting a rise in the pressure above the set profile, signal a pump stop via the SCADA system, and the pressures are then controlled by the DSS using the remaining valves and occasionally pumps starts and stops.

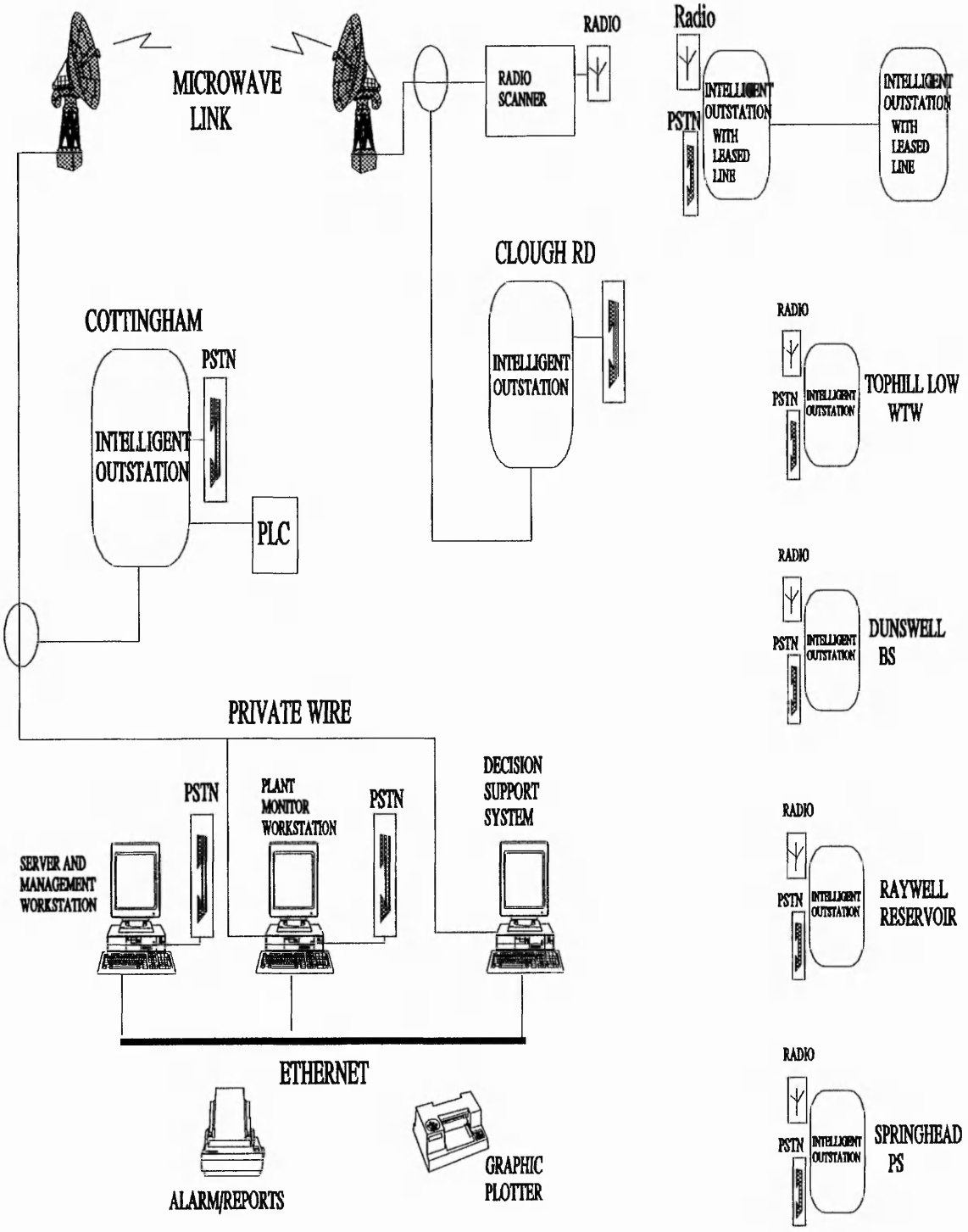

Figure 3: Hull Sources,Scada & Decision Support System.

1.7 Integrating SCADA systems and Hydraulic Models for water distribution control.

The SCADA system is only a part of a distributed control system(DCS). The infra-structure of the SCADA system corresponds to control and monitoring activities within a DCS. However, the decision making process for determining the best course of action while meeting demand at minimum cost, is carried out in another supervisory layer. Integration of this layer and the SCADA system in use yields a DCS.

The integration of the optimization and control system and the Hull city's SCADA system yielding a DSS for the control of the Hull water sources, is one example of DCS. The system is designed to collect data from a SCADA system, decide on the most appropriate course of action, based on meeting demand at minimum cost, while being constrained by distribution pressure and reservoir conditions. Once a control schedule for pumps and valves has been determined, the operating instructions are returned to the SCADA system for action.

The Integrated SCADA simulator (ISS) of DUPage County,Chicago (DWC) in United States, is yet another example of such systems in operation. DWC sought technology to combine current water distribution system data with a hydraulic model for near real-time simulation. The link between DWC's SCADA system and hydraulic model would provide the tool to allow operations personnel to become skilled managers of their water distribution system**[121]**. A computer network (two central processors and five workstations) runs the SCADA system for command and control,of system pumps,valves,

and other essential services. Pressures, flow rates, and equipment status are reported on three-second scan frequencies. The SCADA system, activated in 1991 is now an integral part of the DWC system operation (ISS). The critical success factor for the implementation of the ISS lies in the ability of operators to efficiently use these decision support tools under normal and emergency operating conditions. The capability of the ISS to take current operating data and combine them with projected water demands in near real-time, provides an important pumping and storage management tool.

1.8 Integrating SCADA systems - Future trend.

The use of SCADA systems for automatic control and monitoring of industrial processes (including water distribution systems), has been prevalent since their birth in 1960s. As a result, subsequent developments have followed in areas such as "Programmable Logic Controllers (PLC)" in the early 1970s, Distributed Control Systems (DCSs) in 1975, and PC-based SCADA systems much later**[26,35,109,121]**.

Integrating the current SCADA, DCS, PLC, PC-based systems technologies can provide data acquisition control and management capabilities for local, wide area and global area facilities, whether these be offshore platforms, oil, gas or "water networks", or electric utility distribution networks. Integration of these technologies must however meet the operational and facility management requirements of the end user whether a major offshore platform or a small, low budget SCADA system.

In the case of water distribution systems, for example, the requirements include: (i)

design of the water system infrastructure - the design includes the hydraulic components of the system (i.e. pipes,pumps and reservoirs), (ii) supply of water to consumers - to provide the demanded water quality, with adequate pressure to all consumers in that area, (iii) efficient management of the overall supply and distribution system.

The objective of an efficient computerized control and information management of a water distribution system, includes minimization of high operating cost(e.g. electricity cost for pumping), control of water quality, and leakage control. Furthermore, control of such system is in two fold: first is the "off-line" control which involves the use of computer control modules for network simulation, demand prediction and schedule optimization(i.e. they do not require reference to real-time system measurements), and finally is the "on-line" control of the overall water system operation by accommodating the "real-time" interaction between the telemetry system, the control system and the water system (Fig.4).

The telemetry and SCADA systems have greatly improved the monitoring and control of water systems. The monitored signals are acquired by use of remote transducers and are transmitted directly to control system, whereas the command signals are generated by the control system and transmitted to remote actuators.

Historically, there have been communication data transmission rate limitations on SCADA systems implementation since system economic justifications also included communication facilities. With current and imminent technology implementation of local,

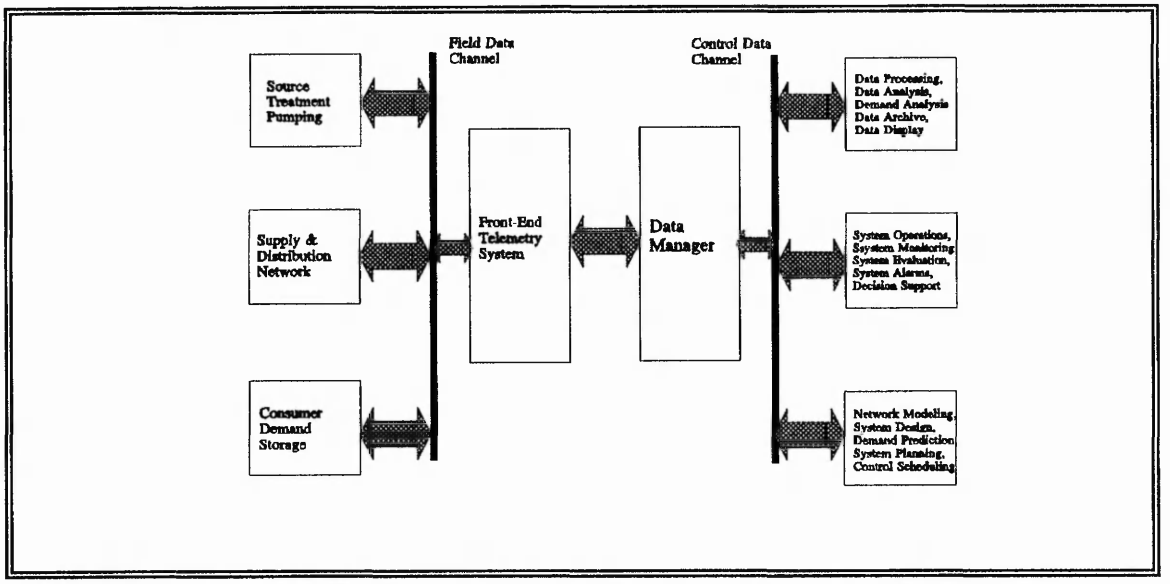

Figure 4: Computer application in water supply and distribution systems.

wide and global area networks based on high data rates fibre optic cables, satellite communications, and ISDN networks, will provide a data highway infrastructure primarily for corporate data transmission requirements but will also include real-time supervisory control and data acquisition. The minimum data transmission rate for such data transfer infrastructure, may be 64kb/s. This 64kb/s data channel may be considered as the narrow lane of the overall data transfer super highway. With a large number of orbiting satellites(ORBCOMM will have 36 in orbit by 1998) there will be the capability to provide truly global communications.

The answer to the question "how is the integration of the aforementioned systems achieved", however, is the term defined by International standard organisation(ISO) as "open system". The open systems through UNIX, X-windows, and Ethernet TCP/IP, also the explosion in low cost processor power has had a huge impact in the process of integrating the aforementioned systems on one control highway. For example, X-windows

is seen by many as the single most important development for bringing together system computing, including DCS as part of management information system(MIS). X-windows are versatile, device independent and supported by wide range of platforms, from mainframes, through workstations to PCs. It is operating system independent, being well supported especially by UNIX variants.

Another factor for easier integration, is object-oriented programming. Objects can be anything from system items to control loops. Once an object is defined, it can be reused or instantiated wherever it is required with minimal effort. For configuration purposes, icons representing objects can be named and linked together, all performed on screen.

The distributed control approach encourages further distribution of intelligence with the distributed system. The ultimate goal is to provide intelligence for sensor diagnostics, monitoring and control with fault tolerance at I/O level. Thus achieving independence from the host. Furthermore, included in their systems some companies employ new technologies such as "neural networks" and "fuzzy process controllers". Although in their infancy, they offer control in areas of non-linear processing and noisy signals, which were difficult to model.

1.9 Computer simulation of water networks.

Simulation of water distribution systems involves the solution of a large set of simultaneous equations. The early water distribution simulation software although

successful in simulating water networks suffered from its inability to deal with network growth which is an inherent feature of water distribution systems. The growth in network size resulted in higher computational time and storage requirements which frequently outpaced the available computing resources. Even with the introduction of sparsity exploiting techniques the computational requirements of the simulation algorithm was quasi-quadratic. Clearly this meant that the use of simulation algorithms on realistic networks would be impractical due to the loss of real-time performance or computational requirements that are not economically justified. A topological model of water networks is constructed from the flow and pressure relationships. An accurate model for the water network behaviour can be formulated by simply applying the basic rules of continuity within pipe networks **[145]**.

This model is comprised of a large set of non-linear relationships depicting the inflows and the outflows at a node or flow between two nodes of the pipe network. Therefore, solving network flow problems involves solving the derived set of simultaneous non-linear equations. Since the equations are non-linear the methods of solution are iterative and limited. Three most commonly used techniques are **[140]**;(i) the Newton_Raphson method, (ii) the linear theory method, and (iii) the Hardy_Cross method, which are explained in detail in the following sections.

1.9.1 Mathematical model of water networks.

A water supply and distribution system consists of a collection of nodes that are interconnected by various elements such as pipes, valves, pumps,and reservoirs. Each

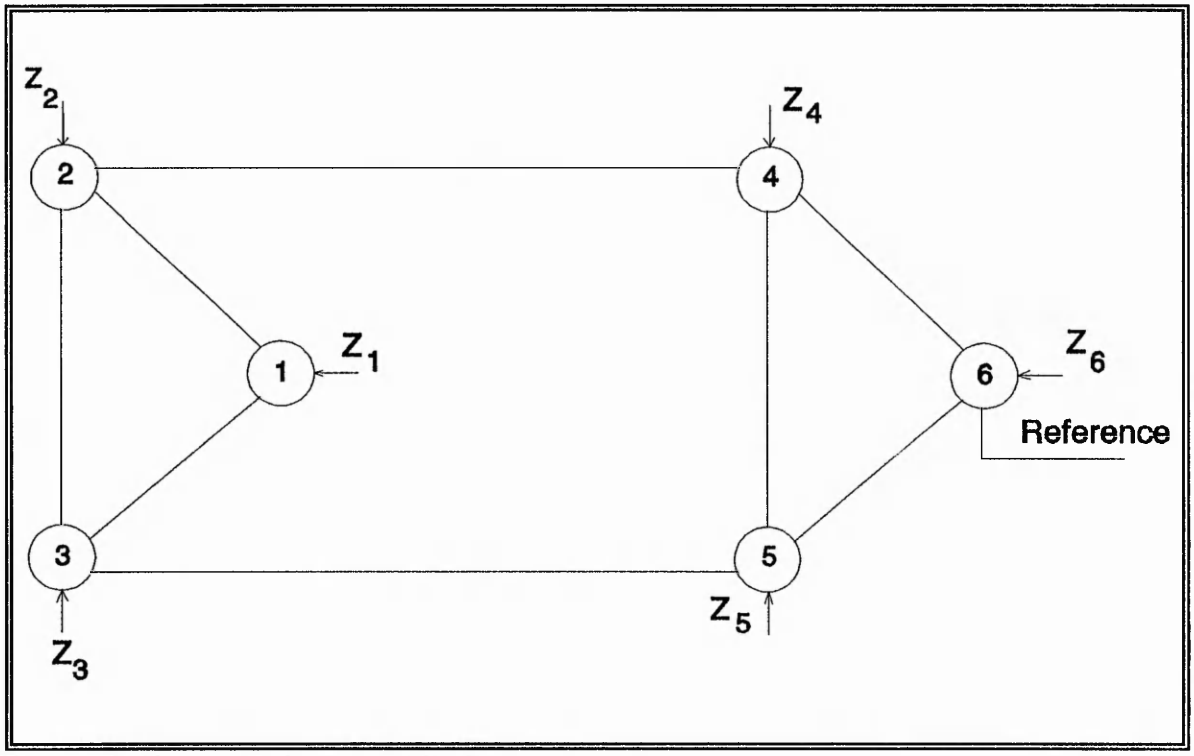

Figure 5: System before partitioning.

element in the network is characterised by a mathematical function that describes the relationship between the element flow and the head difference between the two ends of the element. The form of the relationship depends on the physical characteristics of the element. The system governing equations can be formulated in accordance with the following rules: (i) nodal mass balance_ the algebraic sum of all the inflow and outflow at each node is equal to zero, and (ii) energy conservation - the total sum of the all head losses around any loop in the network is equal to zero.

Using these rules and the functional relationships between flow and pressure drop for every link of the network (Fig.5), the pipe flow f_{ij} equation can be derived which is dependent non_linearly on the pressure difference at the end_nodes of the pipe,

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

where R_{ij} is the hydraulic resistance of the i-j pipe and the vector of the nodal pressures is $x = \{ x_1 \dots x_n \}^T$; n is the number of nodes in the network.

The nodal pressures **x** are usually calculated from mass balance equations in n-1 network nodes and one reference pressure in an arbitrarily selected n-th node.

$$g_i(x) = \sum_{j \in \Omega_i} f_{ij}(x) \quad (2)$$

$$g_n(x) = x_r \quad (3)$$

where $g_i()$ is a mass balance in node i (i.e. i=1..n-1), Ω_i is a set of nodes adjacent to node i, $g_n(x)$ is a pressure measurement in a reference node the value of which is x_r.

In the network without storage elements, $g_i()$ corresponds to the consumption/supply out node i, for $i=\{1,\dots n-1\}$, and to a measurement of the reference pressure in node n.

$$z_i = g_i(x) \quad (4)$$

The system of non_linear equations to be solved can therefore be represented in a compact form as:

$$Z=g(x) \tag{5}$$

where $Z=[Z_1 \ldots Z_{n-1},Z_n]^T$ and $g(x)=[g_1(x) \ldots g_n(x)]^T$.

The solution of (5) involves linearization of the system of equations and iterative improvement of the initial estimate of the vector x , x_0.

$$g(x)=g(x^k)+\frac{\delta g(x)}{\delta x}\Big|_{x_0}\Delta x \tag{6}$$

introducing $J=(\delta g/\delta x)$ for the Jacobian matrix and noting(5),

$$Z=g(x^k)+J\Delta x \tag{7}$$

So

$$\Delta x=J^{-1}(Z-g(x^k)) \tag{8}$$

and the iterative solution is obtained as:

$$x^{k+1}=x^k+\Delta x \tag{9}$$

1.9.2 The Steady-State Network solution techniques.

The non_linearity of the mass balance equations imply that the solution technique is iterative. The three most frequently used methods are; Linear theory, Newton-Raphson and Hardy-Cross method. These methods are presented in the following sections.

LINEAR THEORY METHOD

The linear theory method was initially developed in a loop formulation to determine the set of unknown flows **[145]**. More recently, the method was developed to solve for the nodal heads **[70]**. In both cases, the net inflow/outflow to the network was assumed known, which is perhaps a reasonable assumption at the design stage but is less tenable in the context of operational behaviour.

In a network of N pipes and J nodes and L loops, there are exactly (J-1) linear continuity equations:

$$\sum_{j \varepsilon S} b_{ij} Q_j = Z_i \qquad (10)$$

where Q_j = flow in the jth pipe in loop,

$$b_{ij} = \begin{cases} -1 & inflow \\ +1 & outflow \\ 0 & unconnected \end{cases} \qquad (11)$$

Z_i = consumption at node i,

S=[1,2, ... k], k=N-1 (no. of pipes-1) independent nodal equations.

The L non-linear energy equations are:

$$\sum_{i \varepsilon T} a_{1i} Q_i = \delta h_1 \qquad (12)$$

where T = set of pipes incident with loop 1, where 1=[1,2, ...L]. Collecting eqns.(10) and (12) together a matrix of continuity and energy equation given by eqn.(13):

$$\begin{bmatrix} B \\ A \end{bmatrix} [Q\ ] = \begin{bmatrix} Z \\ \delta h \end{bmatrix} \quad (13)$$

The way to linearize the energy equations is to let Q_i^{n-1} be a constant for a given iteration. The result is to solve N-1+L linear equations with the same number of unknowns. To solve the network problem it is necessary to solve the linear system of equations, recalculate the a terms, and resolve the linear equations repeatedly until the solution converges. The linear theory method tends to overcorrect the Q_i's so that it is possible to base the a terms, not on the new value of the Q_i's but on the weighted average of the old and new Q_i's. This tends to speed convergence.

NEWTON_RAPHSON METHOD

The Newton_Raphson method is a powerful numerical method for solving systems of non-linear equations. This method formulates a set of simultaneous linear equations which can be solved for flow or pressure correction in the water network.

The equation(5) is comprised of a set of nonlinear equations. Since the equations are non_linear the solution method would be iterative. Thus the solution of (5) involves linearization of the equations in (5) and iterative improvement of the initial estimates of the pressures as given by:

$$g(x) = g(x^k) + \frac{\delta g(x)}{\delta x}\Big|_{x_0} \Delta x \quad (14)$$

$$\Delta x = J^{-1}(Z - g(x^k)) \quad (15)$$

and the improvements to the initial estimates found by:

$$x^{k+1} = x^k + \Delta x \quad (16)$$

Convergence is achieved comparatively quickly with this approach since NEWTON_RAPHSON method adjusts the pressures in all the nodes simultaneously. This is particularly important when analyzing networks having large numbers of pipes. Another important factor in achieving faster convergence is the closeness of the initial estimates to the real pressure measurement values, the closer these estimates are to the actual values the faster the convergence.

HARDY_CROSS METHOD

The Hardy_Cross method is one of the first and widely used method of analysis**[145]**. This method makes corrections to initial assumed values by using a first order expansion of the energy equation in terms of correction factor for the flow rate in each loop in the water network. The relevant equations for Hardy Cross method can be derived from eqn.(8) by rearranging it to:

$$J\Delta x = (g(x^k) - Z) \quad (17)$$

This is analogous to solving a set of simultaneous equations(**Ax=b**). Thus, the continuity equations for the pipe network using the loop equations(ΔQ) is:

$$F(\Delta Q_k) = \sum_{i=1}^{m_k} a_i \, |Q_i + \Delta Q_k|^n \qquad (18)$$

where a_i = constant,

Q_i = initial estimates of flow in ith pipe(satisfies continuity _ Known),

m_k = number of pipes in the kth loop(known),

ΔQ = correction to kth loop to achieve convergence(unknown),

F = difference in the head between the two fixed pressure points.

Now by rearranging eqn(18) for the kth loop, we can calculate ΔQ_k.

Reflecting the correction according to the gradient descent for a single equation gives:

$$\Delta Q_k(n+1) = \Delta Q_k(n) - \left(\frac{F(\Delta Q_k)}{\acute{F}(\Delta Q_k)}\right) \qquad (19)$$

This process continues until ($\Delta Q_k(n+1) - \Delta Q_k(n+1) \simeq 0$), when the convergence is achieved. The process is of course iterative and is dependent on the accuracy of the initial guess which must be reasonably good if an answer is to be obtained rapidly. However, the method is suitable for manual solutions and small computers or hand calculators and produce adequate results for most problems. Furthermore, Hardy_Cross method can be viewed as the special case for Newton_Raphson method. The Hardy Cross performs iterations on separate equations, one at a time, while the Newton_Raphson method iterates on the set of equations simultaneously. The Hardy Cross method was developed to facilitate hand computations, and has the advantage of simplicity. The simplicity of the method is helpful in programming the method, but what is more important is the small amount of storage required by it. This is due to the fact that every node is considered

individually and the mass balance equation is formulated and solved for that particular node before other nodes are considered. In contrast, in Newton-Raphson method, the mass-balance equations of the whole network is formed in a system matrix and solved in order to obtain the improvement to the initial estimate of pressures in the system.

The Hardy Cross method suffers from a problem of solvability and convergence **[35]**. Various conditions, such as large pipe diameters or very low flows, which cause the iterative scheme to converge very slowly, or even diverge. Ad hoc procedures have been developed **[35]** to improve the convergence under such conditions, but there is no guarantee of convergence.

1.10 CONCLUSION

1 - The inherent nonlinearity of water distribution systems implies that the on-line monitoring of such systems, is a computationally intensive task. However, since the water distributions networks are topologically comprised of semi-independent systems, the distributed computing scheme seems an ideal match for modelling such systems. The goal of this work was therefore to develop a suitable distributed simulation algorithm and an appropriate distributed computing environment.

2 - Future widespread use of optimal control technology in water supply and distribution systems are likely to be dependent on an increase in the use of more sophisticated SCADA systems and the availability of more commercially available control software.

The present trend is towards the integration of SCADA systems and DCSs, thus bringing about a more effective and efficient control of real-time systems including the water distribution system. In the future, water utilities will continue to benefit from the new on-line decision support software derived from the integration of the aforementioned systems. Enhanced software will yield greater benefits through more effective data management, reduce costs, wider applications and improved staff effectiveness.

3 - The Newton-Raphson method was selected from amongst the methods described in the preceding sections since, firstly there are fewer equations to solve, and secondly, the nodal equations are very much easier to formulate and automatically give maximum sparsity.

CHAPTER 2: Solution techniques for linear system of equations.

2.1 Numerical Solution techniques

The set of simultaneous equations derived from the mathematical model of the water distribution network, for all three methods are equations (8),(13) and (18) and they belong to: "linear theory" method, "Newton_Raphson" method and "Hardy Cross" method respectively. Solution of the derived simultaneous equation is analogous to finding the solution **x** of the system **Ax=b**, where **A** is a nonsingular square n by n real sparse matrix and **b** a full vector. The algorithm may be grouped into two categories: direct methods and iterative methods **[33]**. Direct methods are based on Gauss elimination; the equations or the unknowns of the system are modified in successive steps until the solution is found. In the iterative methods, an initial guess is usually made for **x**, and this guess is then improved until sufficient accuracy is obtained. Both methods have advantages and disadvantages in each particular case and it is difficult to state general rules as to which is the most convenient. This chapter focuses on direct methods for solving **Ax=b**, with **A** real nonsingular square matrix of order **n**.

2.1.1 Gaussian Elimination Technique

Gauss elimination is a well known procedure for solving linear equations **[43,50,52]**. The elimination by columns is the most popular version of the algorithm. In this approach diagonal elements are chosen as pivots in the same order as they appear in the main diagonal. Thus, it is important to order matrix **A** so that the elements on the main diagonal are not only greater than zero but also results in having a diagonally dominant matrix A (matrix is diagonally dominant by rows if each diagonal element is not less than

the sum of the moduli of the other elements in its row). Diagonal dominance of **A** would help to minimize the round_off error, thus resulting in a more stable system numerically. The topic under which these issues are studied is called pivots selection or "ordering" which shall not be discussed any further here.

Considering the system **Ax=b**, Gaussian elimination by columns consists of n steps. The purpose of the kth step is to eliminate all the nonzero elements of the matrix which lie on column K below the diagonal. At the first step, the nonzeros of column1 of A are eliminated by subtracting convenient multiples of row1, element by element, from each of the remaining rows with a nonzero in column1. The element A_{11} belonging to the row that is going to be subtracted from other rows(row1 in this case) and to the column that will be eliminated(column1 in this case), is called the pivot and assumed to be nonzero. Prior to elimination row1 is normalized by dividing all its nonzero elements by the pivot. A matrix $A^{(2)}$is obtained with $A_{i1}^{(2)}=0$ for $i>1$ and $A_{11}^{(2)}=1$.

At the second step, $A_{22}^{(2)}$ is selected to be the pivot. Again we assume $A_{22}^{(2)}\neq 0$. Row2 is normalized and all nonzeros of the second column below the diagonal are eliminated by subtraction of convenient multiples of the normalized second row from the corresponding rows. Note that, since $A_{21}^{(2)}=0$, the elements of column 1 will not be affected. A matrix $A^{(3)}$ is obtained with $A_{i1}^{(3)}=0$ for $i>1$, $A_{i2}^{(3)}=0$ for $i>2$ and $A_{11}^{(3)}=A_{22}^{(3)}=1$. In another words, $A^{(3)}$ is upper triangular unit diagonal in its first two columns.

At the beginning of the kth step we have a matrix $A^{(k)}$ with zeros on its first k-1 columns below the diagonal and ones on the k-1 initial positions of the diagonal. The

following

example shows $A^{(k)}$ for the case n=6, k=3:

$$A^{(3)} = \begin{bmatrix} 1 & . & . & . & . & . \\ 0 & 1 & . & . & . & . \\ 0 & 0 & . & . & . & . \\ 0 & 0 & . & . & . & . \\ 0 & 0 & . & . & . & . \\ 0 & 0 & . & . & . & . \end{bmatrix} \quad (20)$$

This process continues until, at the end of step n the matrix $A^{(n+1)}$ is obtained, which has only zeros below the diagonal and ones on the diagonal, and is thus upper triangular unit diagonal. Thus, the kth step of Gaussian elimination by columns is equivalent to pre_multiplication of $A^{(k)}$ by the inverses of elementary matrices D_k and L_k^c:

$$A^{(k+1)} = (L_k^c)^{-1} D_k^{-1} A^k \quad (21)$$

where

$$(D_k)_{kk} = A_{kk}^{(k)} \quad (22)$$

$$(L_k^c)_{ik} = A_{ik}^k \quad (23)$$

for i>k, and $A^{(1)}=A$ (D_k is a diagonal elementary matrix and L_k^c is a lower column elementary matrix).

2.1.2 Gaussian_Jordan elimination technique

The algorithm for Gauss_Jordan elimination **[53]** is similar to Gaussian elimination, the main difference being that, at the beginning of step k, the matrix $A^{(k)}$has zeros in its column 1 to k-1 both above and below the diagonal. The following example shows $A^{(k)}$for the case n=6,k=3:

$$A^{(3)} = \begin{bmatrix} 1 & 0 & . & . & . & . \\ 0 & 1 & . & . & . & . \\ 0 & 0 & . & . & . & . \\ 0 & 0 & . & . & . & . \\ 0 & 0 & . & . & . & . \\ 0 & 0 & . & . & . & . \end{bmatrix} \qquad (24)$$

The kth step consists of the elimination of the nonzeros on the column K of $A^{(k)}$ both above and below the diagonal. Row k is first normalized by dividing all its elements by the diagonal. Row k is first normalized by dividing all its elements by the diagonal element. Then, convenient multiples of the normalized row k are subtracted from all those rows which have a nonzero on column k either above or below the diagonal. The matrix $A^{(k+1)}$ is thus obtained with zeros on its k initial columns. This process is continued until, at the end of step n, the identity matrix $A^{(n+1)}(\equiv I)$ is obtained. The kth step of Gauss_Jordan elimination by columns is equivalent to pre_multiplication of $A^{(k)}$ by D_k^{-1} and by the complete column elementary matrix $(T_k^c)^{-1}$.

$$A^{(k+1)} = (T_k^c)^{-1} D_k^{-1} A^k \qquad (25)$$

where $A^{(1)} \equiv A$ and :

$(D_k)_{kk} = A_{kk}^{(k)}$

$(T_k^c)_{ik} = A_{ik}^{(k)}$ for all i/=k.

Thus, we have:

$$(T_n^c)^{-1}D_n^{-1}\ldots(T_2^c)^{-1}D_2^{-1}(T_1^c)^{-1}D_1^{-1}A = I \qquad (26)$$

2.2 Matrix Factorization techniques

The analysis of a large network systems involves the solution of large number of simultaneous equations of the form **Ax=b [100,107,133]**. Furthermore, several solutions are often required with the same coefficient matrix A but with a series of different b vector. The solution of the preceding set of linear equations can be written as $\mathbf{x}=\mathbf{A}^{-1}\mathbf{b}$, however, explicit inversion of A, though A is sparse, results in a dense A^{-1}. This means an increase of n^2 in the storage requirement and n^3 increase in the arithmetic operations.

An alternative method is Gauss elimination(or methods based on Gauss elimination). As discussed in section 2.1, this method reduces the number of arithmetic operations to about $(n^3/3)$, but requires an indeterminate number of storage locations. In general however it is significantly better than direct inversion, but requires a systematic form of logic to achieve an efficient computer program. This is achieved using one of the various modifications of the basic Gauss elimination techniques, which are generally known as matrix factorization methods. These methods use Gauss elimination to obtain the inverse of the coefficient matrix implicitly as the product of several factor matrices. They do not in themselves improve on the storage requirements of the number of arithmetic operations need using Gauss elimination. However, because of their systematised logic, they lend themselves to numerical techniques and computer programming, which, when sparsity

techniques are included, can drastically reduce both the number of operations and storage requirements.

2.2.1 Triangulation of Matrices

Triangular decomposition(or triangulation of matrices) is one of the most widely used methods of manipulating coefficient matrices to solve simultaneous linear equations. The Triangular decomposition of the coefficient matrices is performed in two phases: i) forward elimination (the implicit factorization of A into the product of a lower triangular(L) matrix and upper triangular matrix(U)) and ii) back substitution(solving the upper triangular system for the unknowns, x vector) **[33]**.

The LU method of factorization consists of expressing the coefficient matrix A as the product of two factor matrices, such that:

$$A=LU \tag{27}$$

where L= a lower triangular matrix,

and U= an upper triangular matrix which has unity elements on its diagonal.

The ability to factorize in this way is the fundamental property of any square matrix. Thus if the set of simultaneous linear equations to be solved are written in matrix form **Ax=b**, then substituting for A(eqn 19) gives:

$$LUx=b \tag{28}$$

Letting $Ux=y$, then from equation 20, we have:

$$Ly=b \qquad (29)$$

Since L is lower triangular matrix, y can be found from L and b by forward substitution, and since U is an upper triangular matrix the unknown vector x can be found from U and y by forward substitution.

To improve numerical stability, some form of numerical pivoting is usually required at each factorization step. Numerical pivoting attempts to reduce round_off error by reducing growth in the magnitude of the matrix elements**[33]**. There are three basic methods with varying degree of numerical stability: i) firstly is the "complete pivoting", this method selects the element $a_{r,c}$(of matrix A) with the largest absolute value in the ith submatrix($A_{x,y}$, $i \le x \le n$, $i \le y \le n$) and interchanges rows i and r and columns i and c. It is the most stable method, but it is rarely used since it involves n^2 search for each pivot elements, ii) secondly is the partial pivoting which is the most commonly used technique. It selects the element $a_{r,i}$ with the largest absolute value in the ith column($A_{x,i}$, $i \le x \le n$) and interchanges rows i and r. It is not as stable as the complete pivoting in theoretical sense, but in practice gives good results**[143]**. It requires only an n search for each pivot element, iii) thirdly is the pairwise pivoting. This method selects between a pair of rows, using one to reduce the other. The left most nonzero column on the row pair is examined and the element with the largest absolute value is selected as the pivot. Although pairwise pivoting is theoretically less stable than partial pivoting, in practice it appears to be just as stable **[128]**. Pairwise pivoting has been used in a parallel algorithm for solving dense

system of linear equations **[116]**.

2.2.2 Bi_factorization technique

The bi_factorization(BF) method is a combination of the well known product form of inverse and triangular factorization. The original factorization method was modified by Tinney and other authors **[36,133,134]**. The Zolenkopf's BF **[110]** is a derivation of Tinney's approach. The principle requirement of the BF method is that, it should be used for sparse matrices that have nonzero diagonal terms and are strictly symmetric or asymmetric in element value but with a symmetric sparsity structure. Some may argue that this is a disadvantage since there are many problems which do not have these features, there are conversely many systems that do. The BF method is an important and frequently used technique for solving large engineering problems.

The method is based on finding 2n factor matrices for an nth order problem, such that the product of these factor matrices satisfies the requirement:

$$L^{(n)}L^{(n-1)}\ldots L^{(2)}L^{(1)}AR^{(1)}R^{(2)}\ldots R^{(n-1)}R^{(n)} = U \qquad (30)$$

where A = original coefficient matrix,

L = left_hand factor matrices,

R = Right_hand factor matrices,

U = Unit matrix of order n.

Premultiplying equation 22 by the inverses of the left_hand factor matrices consecutively gives:

$$AR^{(1)}R^{(2)}\ldots R^{(n-1)}R^{(n)} = (L^{(1)})^{-1}(L^{(2)})^{-1}\ldots(L^{(n-1)})^{-1}(L^{(n)})^{-1} \quad (31)$$

Post_multiplying equation 23 by left_hand factor matrices consecutively gives:

$$AR^{(1)}R^{(2)}\ldots R^{(n-1)}R^{(n)}L^{(n)}L^{(n-1)}\ldots L^{(2)}L^{(1)} = U \quad (32)$$

Finally, premultiplying equation 24 by A^{-1} gives:

$$R^{(1)}R^{(2)}\ldots R^{(n-1)}R^{(n)}L^{(n)}L^{(n-1)}\ldots L^{(2)}L^{(1)} = A^{-1} \quad (33)$$

Thus, the factor matrices L and R at the kth reduction step are given by:

$$L_{kk}^{(k)} = \frac{1}{a_{kk}^{k-1}} \quad (34)$$

$$L_{ik}^{(k)} = -\frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} \quad (35)$$

$$R_{(kj)}^{(k)} = -\frac{a_{kj}^{(k-1)}}{a_{kk}^{(k-1)}} \quad (36)$$

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \frac{(a_{ik}^{(k-1)}a_{kj}^{(k-1)})}{a_{kk}^{(k-1)}} \quad (37)$$

For symmetrical matrix A:

$a_{ik}^{(k-1)} = a_{ki}^{(k-1)}$

therefore $R_{ik}^{(k)} = L_{ki}^{(k)}$.

In the case of symmetrical coefficient matrices, equation $R_{ik}^{(k)} = L_{ik}^{(k)}$ indicates that, except for the diagonal elements , the kth row of $R^{(k)}$ is identical to the kth column of $L^{(k)}$. Also, the diagonal elements of $R^{(k)}$ are all unity and since these are known implicitly, it is sufficient only to evaluate the elements of $L^{(k)}$. Therefore, the required number of operations and the amount of storage is reduced to almost a half.

2.3 Sparse Matrix Technology

Introduction

Many of the elimination or factorization techniques described in the preceding sections, used in the solution of a system of linear equations of the form **Ax=b**(A is a sparse n*n matrix) are not the most efficient way of solving such sparse systems. These techniques do not take advantage of the sparsity that is present in matrix A and its factored forms. Another unfortunate property of the elimination or factorization techniques is that new nonzero elements can continuously be generated(Fill_ins). This would further degrade the original system's degree of sparsity. When using direct methods for the solution of sparse linear equations, it is important to design the algorithms to preserve as much as possible of the system's initial sparsity. This would be advantageous when considering sparse systems, the most evident is in information storage and retrieval systems. Algorithms can be made more efficient if only the nonzero elements with matrix A and its factored forms are stored and processed. Furthermore the fill_in phenomenon can be controlled by the

order in which the pivotal rows and columns are selected.In the following sections the sparsity exploiting techniques that are employed in the solution of a set of linear equations derived from the water distribution system, will be studied.

This section presents the solution methods employed in the simulation of large water distribution systems. The solution techniques are then compared with all other existing solution techniques in areas such as "pivot selection", "sorting and ordering" and "storage".

2.3.1 Sparsity_Directed Elimination Techniques

In the simulation of water networks, the mathematical model derived for the system incorporates the **MA28** routine of the Harwell Library **[37]**. This subroutine implements the **Bartels_Golub** decomposition of the system's coefficient matrix. What follows is the description of the algorithm.

For solving a large sparse system of linear equations, represented as **Ax=b**, where A is a large sparse coefficient matrix, and may be unsymmetric. The MA28 routine solves the system by first reordering the matrix into block upper triangular form (Fig.6). This is done by the application of "row" and "column" interchanges to the original basis of Gaussian elimination. The elimination can be defined in the equation form as:

$$M_i M_{i-1} \ldots M_1 A = PUQ \qquad (38)$$

where M_i= is a matrix which differs from I in just one off_diagonal element(representing row operation)

for i={1 ... r},

P,Q = are permutation matrices,

and U = is an upper triangular matrix.

$$PAQ = \begin{vmatrix} A_{11} & A_{12} & \cdots & A_{1r} \\ & A_{22} & & \vdots \\ & & \ddots & \vdots \\ 0 & & & A_{rr} \end{vmatrix}$$

Figure 6: Block Upper Triangular form of matrix A.

Equation Ax=b is easy to solve since the factorization (30) allows A^{-1} to be expressed in the form:

$$A^{-1} = Q^T U^{-1} P^T M_r M_{r-1} \ldots M_1 \qquad (39)$$

Another similar approach , was purposed by Gill and Murray **[51]**. Their method incorporates orthogonal matrices in contrast to Gaussian elimination. Later on Saunders **[120]**adapted Gill and Murray's algorithm for sparse matrices.Basically, it involves using the factorization:

where L = is a lower_triangular matrix,

$$A=LQ \tag{40}$$

and Q = is an orthogonal matrix.

However, the algorithm only stores L, this is explained by the fact that:

$$A^{-1} = A^{T}L^{-T}L^{-1} \tag{41}$$

This can be proved by the following equations:

from (32) $A^{-1}= (LQ)^{-1}= Q^{-1}L^{-1}$,

and $A^{T}= (LQ)^{T}= Q^{T}L^{T}$,

then $Q^{T}= A^{T}L^{-T}$.

Remembering that Q is orthogonal, and substituting in $Q^{T}= A^{T}L^{-T}$ into the equation for A^{-1} above gives:

$$A^{-1}=A^{T}L^{-T}L^{-1}$$

Since orthogonal reduction causes more fill_in than Gaussian elimination, matrix L will have more nonzeros than there are matrices M_i of Bartel_Golub algorithm, but the fact that Q does not have to be stored is a very appealing prospect.

The Forrest and Tomlin approach **[46]** differs from Bartel_Golub in that the permutation is applied to the rows as well as the columns to produce an upper triangular matrix. Unlike Bartel_Golub algorithm, Forrest and Tomlin algorithm performs interchange which results in a greater fill_in and instability**[112]**. This means that the Forrest_Tomlin

algorithm require frequent factorization. However, to remedy the instability problem it is required to monitor the growth in the size of matrix elements and perform factorization when necessary.

Another method was purposed by Golub and later improved by Gentelmen **[53,48]** for the solution the aforementioned system of linear equations with A being m by n where $m \geq n$. Golub suggested the use of orthogonal reduction to upper triangular form. In Golub's algorithm matrix A is decomposed to:

$$A = Q_n U_n \tag{42}$$

where Q_n is an m by n orthogonal matrix(i.e $Q_n^T = Q_n^{-1}$), and U_n is an n by n upper triangular matrix. Thus multiplying both sides of equation(35) by A^T gives:

$$A^T A x = A^T b \tag{43}$$

Substituting (35) into (36) results in:

$$U_n^T Q_n^T Q_n U_n x = U_n^T Q_n^T b \tag{44}$$

which then reduces to(i.e with respect to orthogonality of Qn):

$$U_n x = Q_n^T b \tag{45}$$

and this may be solved by a simple back substitution. The decomposition of matrix A in (35) is performed using **Householder Transformation matrices**. Orthogonal reduction techniques includes the traditional methods of Givens**[52]** and Householder**[62]**.

Peter and Wilkison **[103]** suggest yet another technique based on Gaussian elimination which removes much of the instability present in the method of normal equations. They first perform the decomposition **A=LU**, so that the normal equations take the form :

$$U^T L^T L U x = U^T L^T b \qquad (46)$$

This then reduces to :

$$L^T L U x = L^T b \qquad (47)$$

This may be solved by using the symmetric decomposition:

$$L^T L = L_2 D_2 L_2^T \qquad (48)$$

of the n by n matrix L^TL. The ill_conditioning of the normal equations is avoided by the multiplication of U^{-T} to (39).

Augmented matrix method was proposed by Hachtel**[104]**. Considering the set of equations

$$\begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix} \qquad (49)$$

and applying block elimination the equivalent system:

is obtained from which it is evident that x is our required solution and r is the residual

$$\begin{bmatrix} I & A \\ 0 & -A^T A \end{bmatrix} \begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ -A^T b \end{bmatrix} \quad (50)$$

vector b-Ax. Hachtel's suggestion is to solve (42) directly, taking full advantage of its sparseness and form. This way of expressing the least squares problem was used by Bjorck**[105]** in his iterative refinement method and by Seigel**[123]** to avoid instability problems. Such form of state estimation equations were used in the context of water distribution systems by Bargiela**[12,7]**.

Duff and Reid**[124]** decided however, to ignore symmetry of (42) and use straight unsymmetric Markowitz in choosing the pivot. In this way account can be take of the sparsity of the right hand side and the fact that only a partial solution of (42) is required. The gains from doing this are often so great that they outweigh the disadvantages of utilizing an unsymmetric solution process. this particularly true if it is desired to solve several systems with the same coefficient matrix.

2.3.2 Sparsity Directed ordering and storage techniques

Ordering techniques:

In an ideal situation it would be desirable to solve the system globally and find an ordering which among all the orderings, produces the optimal one in some well defined sense. However, finding an optimum ordering that minimizes the memory requirement and computation time for the solution process is too complex to be achieved for practical problems. Rose and Tarjan**[113]** showed that the problem of finding

and ordering that results in minimum fill_in during an associated Gaussian elimination process is NP_complete, implying that the problem is computationally intractable. Since the memory requirement for solving the system Ax=b is a linear function of the number of fill_ins generated during Gaussian elimination process the results of Rose and Tarjan imply that even a simplified problem of finding an optimum ordering that minimizes memory requirement is computationally a very hard problem. It is very important to realize that, although the word optimum is often used to describe ordering algorithms**[63,135]**, no known algorithm are optimal in a global sense for general sparse matrices. On the other hand a suboptimal ordering method is achievable and purposed by several authors**[63,17,136,75,34,38]**. These techniques differ from one another in one or more of the following: i) the set of elements from which the choice of pivot is made, ii) the rule for selecting a pivot for a given stage of the Gaussian elimination process from a set of candidates.

The objectives of a good suboptimal ordering algorithm are that it must lead to a significant reduction in the number of fill\-ins, arithmetical operations and rounding errors(however, these techniques are heuristics). The two existing ordering algorithms are "a priori" and "local strategy" methods. In the a priori method used by **[38,39,40,41,96,97,86]**, the columns(rows) are first ordered and then, at each stage of the elimination, the pivot is chosen from within the first column of the reduced submatrix A^k shown in Fig.7. Even though these a priori methods give results which are far from optimal, they give a great improvement over not ordering for sparsity at all. Indeed using a crude a priori approximation of Markowitz's**[86]** ordering, Sato and Tinney**[118]**

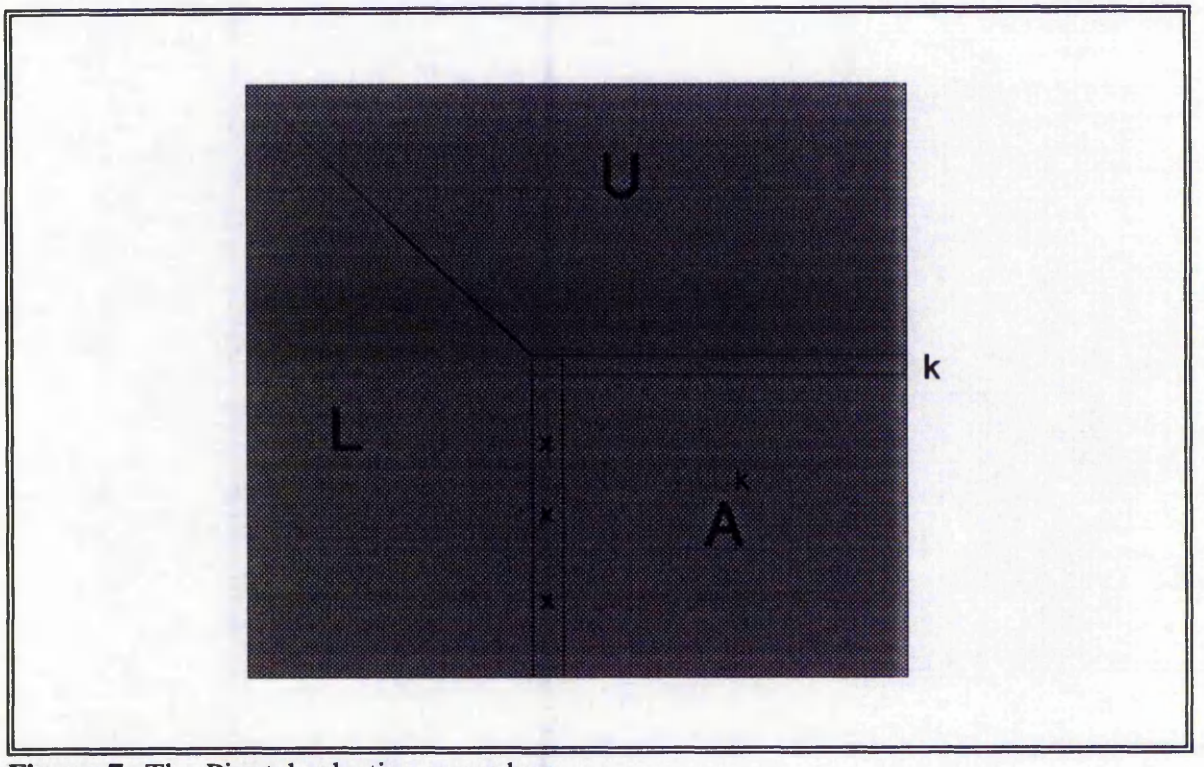

Figure 7: The Pivotal selection procedure.

observe a 4:1 improvement over not pivoting for sparsity.

In local strategies**[13,38]**, the pivot is selected from among all the nonzeros in the reduced submatrix using knowledge of its actual updated structure at that stage of the elimination. The most popular ordering technique is that suggested by Markowitz**[86]** which chooses the nonzeros at each stage, that minimizes the product of the number of other nonzeros in the candidates row and column.

This product is the maximum fill_in that can be created by the $a_{ki}^{(k)}$ pivot element of matrix $A^{(k)}$. Other methods derived by other authors **[13,64,63,38,92,65,66,20,44,14,115,142]** have failed in their attempt to out perform the Markowitz method. On the contrary, the results of the experiments carried out

by**[63,17,136,75,34,38]**show the Markowitz criterion to be about the best.

Markowitz method was adapted the MA28 Harwell subroutine, for its numerical stability. The stability criterion $|a_{kk}^{(k)}| > u.\max_i|a_{ik}^{(k)}|$ is placed on the pivot; an element is rejected if its absolute value is smaller than u times the maximum absolute value of the other nonzero elements in its same row. If u=1, then partial pivoting is performed, while u=0 causes pivots with minimum upper bound on the fill_in to be selected. Gill and Murray**[50]** alleviated the problem of factorized form of A containing more nonzeros than A itself, by avoiding the storage of the orthogonal matrix Q(of A=LQ). Pivotal column was chosen from columns with least nonzeros. A pivot was selected from the element within the pivotal column that had least number of nonzero in its row. It is a stable algorithm for moderately ill_conditioned problems, but not for problems with widely differing row scallings. However, the Golub_Householder and the Golub_Givens procedures are very stable numerically. Powel and Reid**[108]** showed that Householder variant is stable even in the presence of widely differing row scaling, provided that suitable row and column interchanges were included.

The Peter and Wilkison**[106]** algorithm for the ill_conditioned system of equations is as stable as the Golub method as far as stability is concerned. This level of stability is achieved because matrix L^TL(of equation(31)) is well conditioned. This will usually be so if the row interchanges are included in the decomposition (31) to limit the size of the off_diagonal elements of L. From the sparsity point of view this algorithm is usually more satisfactory than the Golub algorithm**[54]**. Following the unconventional approach of

Duff and Reid[37] of the unsymmetric solution of the symmetric problem, it was recommended by Duff[42] that, for the least square problems, as the matrix becomes squarer and more dense, the augmented matrix method becomes a very suitable method for solving these problems. Duff and Reid[43] recommended that, the choice of method should depend principally on the degree of stability that is required. In such cases therefore, they recommended Peters_Wilkinson algorithm. In other cases however, Hachtel's scheme in its symmetric version unless rapid processing of further vectors b is important in which case the unsymmetric approach is recommended.

Storage Techniques

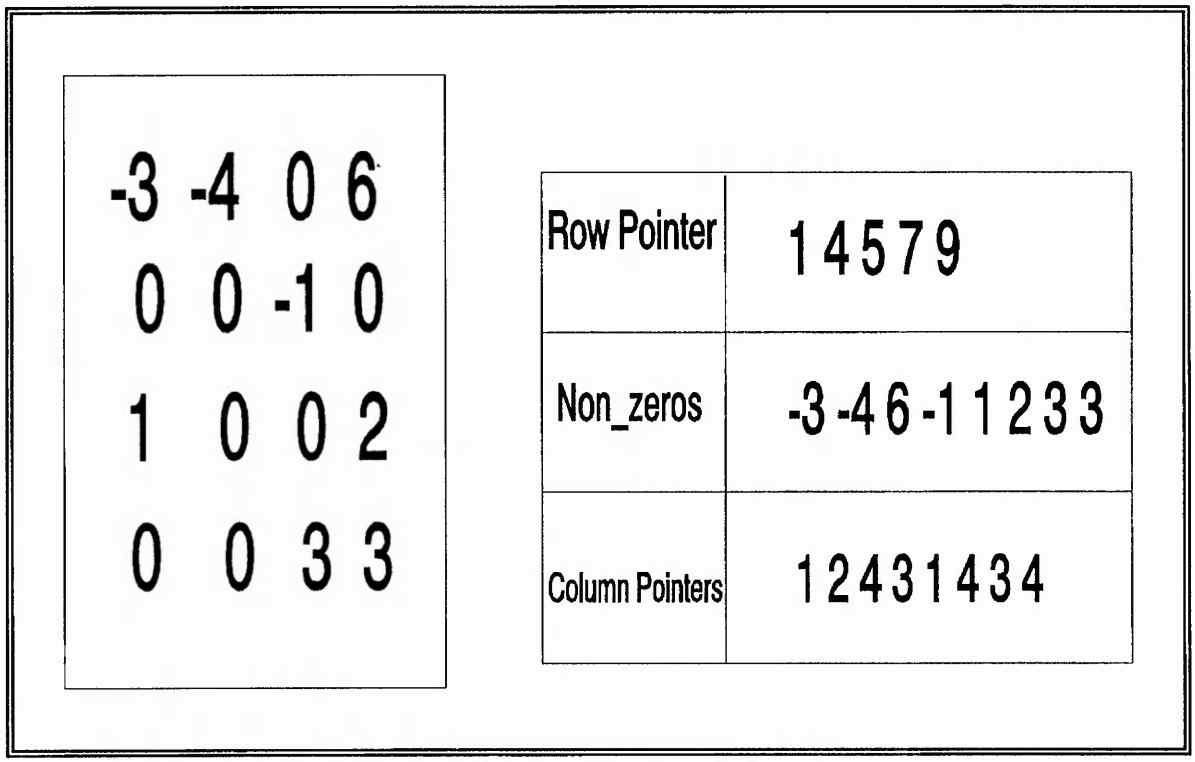

Figure 8: An example of Sparsity_directed storage technique.

The basic requirement of any sparse storage scheme is, easy acquirement of the matrix

elements and minimization of storage requirement for the sparse matrix. However, the best scheme in any particular case is dependent on the structure involved and the use to which the matrix will be put. Primarily storage schemes belong to two main categories:i) static storage scheme(individual elements are merely accessed), and ii) dynamic storage scheme(allowing the change if a zero element to a nonzero element). The simplest and most straight forward of static storage scheme is the coordinate storage used by Phillips**[107]**, Page and Wilson**[98]**. Here, the nonzero elements along with their coordinates are stored. Scheme used by Curtis and Reid**[99,100]** and Brandon**[101]** is called a column pointer/row index(row pointer/column index) scheme and is shown in Fig.8. Another static storage is the bit map used by Gustavson et. al.**[102]**. This method has the advantage of requiring only one bit per element but has the disadvantage of storing bit information for the zeros also. Furthermore, high degree of granularity is required in programming such scheme in high level languages, thus making it unpopular, although it may be suitable for parallel computing machines. However, attributes such as ease of access and economy of space makes the static storage the row pointer/column index(column pointer/row index) the best.

The most important form of dynamic storage scheme is the infamous "link_list"**[79]**. The link_list is consisted of a set of parallel array retaining information such as: i)row index ii)column index, iii) pointer to the next element in the row,iv) pointer to the next element in the column, v) pointer to previous element in the row, vi)pointer to the previous element in the column. However,link_lists usually retain some of these attributes. For example, subroutines MA28 and MA18 use only attributes (iii) and (iv).

To manipulate in this network of interlinked arrays one or two extra integer arrays are required. This is most prevalent in the Bi_factorization technique**[146]**. Gustavson**[55]** has purposed a storage scheme which is used in the Harwell sparse linear programming routine LA05A**[56]**. This scheme essentially store two copies of the matrix: one oriented by rows and the other by columns.

CHAPTER 3: Network Decomposition Techniques

3.1 Introduction

In the preceding chapter, various sparsity directed solution techniques have been discussed assuming that the whole system is analyzed simultaneously. For every network problem, a combination of these elimination and ordering technique can be used to obtain the solution. However, in certain very large sparse network problems these techniques may fail to deliver the required solution efficiently. For example, any change to the physical size of the network would require that the whole system of nonlinear equations be solved all over again. In the case of the decomposed system solution what is required is the solution to the modified part of the network followed by the coordination of solutions. The associated attributes of such techniques are i) reduction of the highly complex and interconnected networks to a set of subsystems each of which require less storage and are generally much easier to solve, ii) they offer better configuration flexibility, that is the ability to simulate water networks of varying size without the incursion of time penalties or the need for different computational techniques in order to cope with the numerical complexity of the problem, iii) greater robustness of the system(i.e. where a processor suddenly fails, its job can be reallocated to the other processor within the distributed processing network). The need for the decomposition techniques as applied to the solution of electrical power distribution systems, has been raised by Kron**[80]** and Himmelblaun**[59]**, Kulikowski**[81]**, Dantzig**[29]**, Nemhauser**[94]**, Lasdon**[83]**, Geoffrion**[49]**, and Mine**[90]**, Mesarovic**[88]** and Vichenevetsky**[137]**.

3.2 Physical structure and sparsity of large water distribution system

Decomposition techniques are particularly suitable for networks which contain groups of highly interconnected nodes, the individual groups being only sparsely connected. Water distribution system is one such system. Each densely populated or industrial area will have a number of nodes interconnected by many pipes , where as different areas will only be connected by a few pipes. Fig.9 shows a 130 nodes water distribution system together with the sparsity pattern of its Jacobian matrix. This is the type of structure which is amenable to decomposition and in particular the network tearing method of Kron which will be described in the following sections. The idea of decomposing such systems is depicted in Fig.10. Because the sparsity of the matrix reflects the sparsity of the original network (provided the rows and columns are ordered suitably) the tearing of the matrix equations can be made by inspecting the graph of the original water network. However, partitioning such systems by inspection may not always result in the best possible decomposed format. Because of this reason, an automatic clustering algorithm has been implemented to partition the network efficiently. This is discussed in Chapter 5.

3.3 Constructing a Decomposed Model

The decomposed model for the system is constructed using the goal coordination principle given by Mesarovic et al**[88]**. The interconnection is cut at the subsystem level and the model itself resembles Fig.11. The model consists of a coordination level and a subsystem solution level. The coordination level must therefore attempt to neutralize the effect of the imbalance incurred due to tearing of the interconnecting elements. The derived subsystems are solved totally independently from each other and the final solution

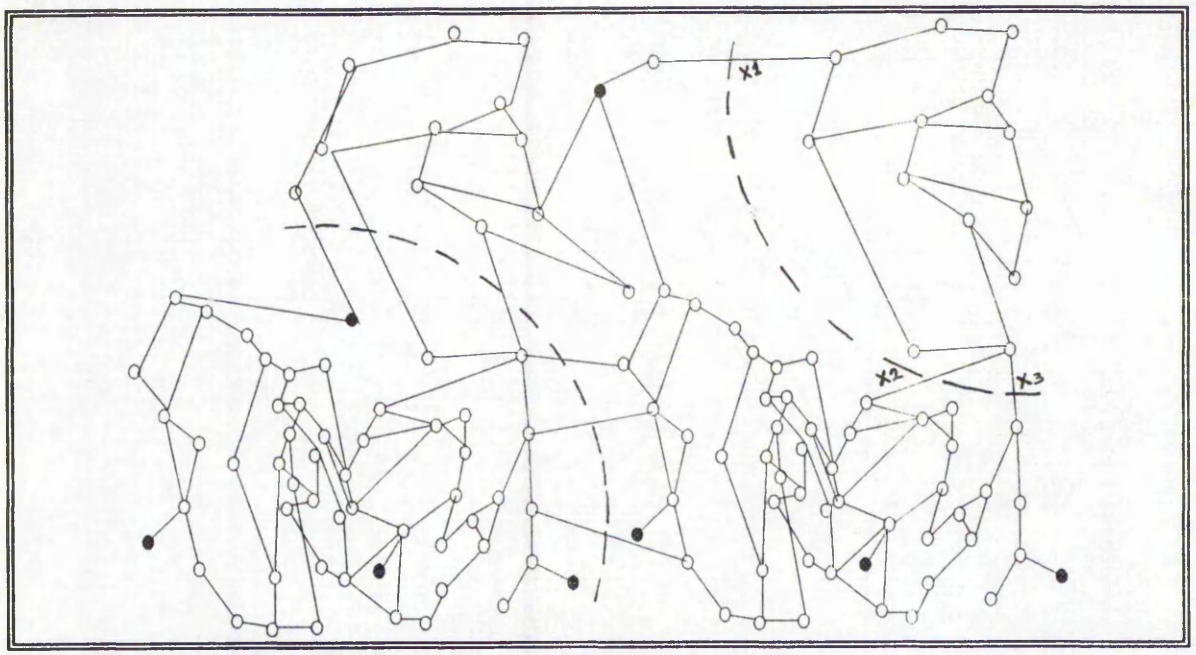

Figure 9: The partitioned 130 nodes network example.

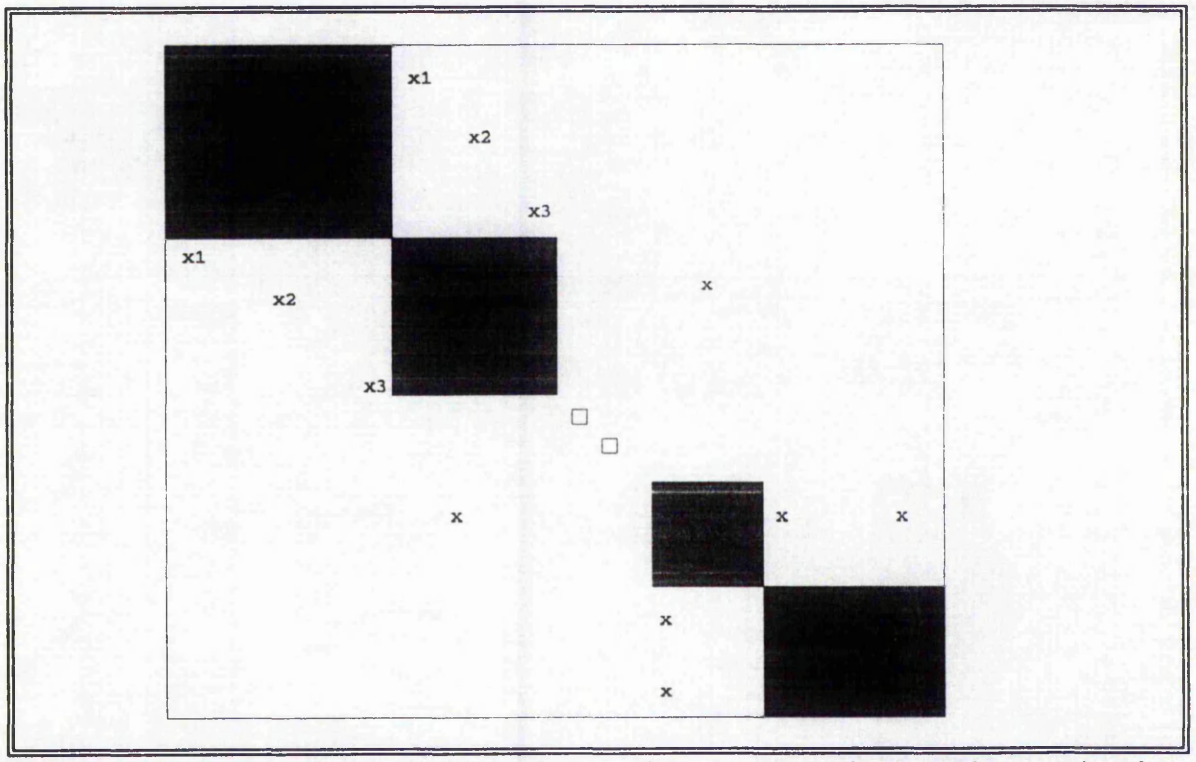

Figure 10: This diagram represents the format of the corresponding Jacobian matrix of the water network shown in Fig.9.

of the system is given by the coordinating level.

3.4 Network Tearing technique

3.4.1 Introduction

Network tearing(Diakoptics) was pioneered by Kron**[80]** and further developed by Happ**[57]**. Roth**[114]**, R.Onederra**[95]** and Shun ichi Amari**[3]** have been able to derive the same equations as Kron. Kron showed that splitting up a system into a number of parts, and solving the problem on each part separately, it is possible for linear systems to combine the subsystem solutions into an overall solution as an analytical process. This results in saving both of computational time and storage space compared to either a distributed iterative improvement or centralised methods. Our study of Kron's network tearing stem from Brameller's**[18]** interpretation of the algorithm. Brameller uses the concepts of circuit topology and elementary matrix algebra in his approach to derive the same equations as Kron. Network tearing involves dividing the original network into a number of component networks so that each subnetwork is isolated from the rest of the system. The matrix of the coefficients for each small network is solved independently, as if the other component networks were non_existent. The solution of the full network is then obtained from the solution of the component networks by a simple routine procedure. The solution obtained is an exact solution within the limitation imposed by the accuracy of the parameters of the original problem and the number of digits used in the calculation. No approximation or iterations are carried out at any stage. Kron's method would

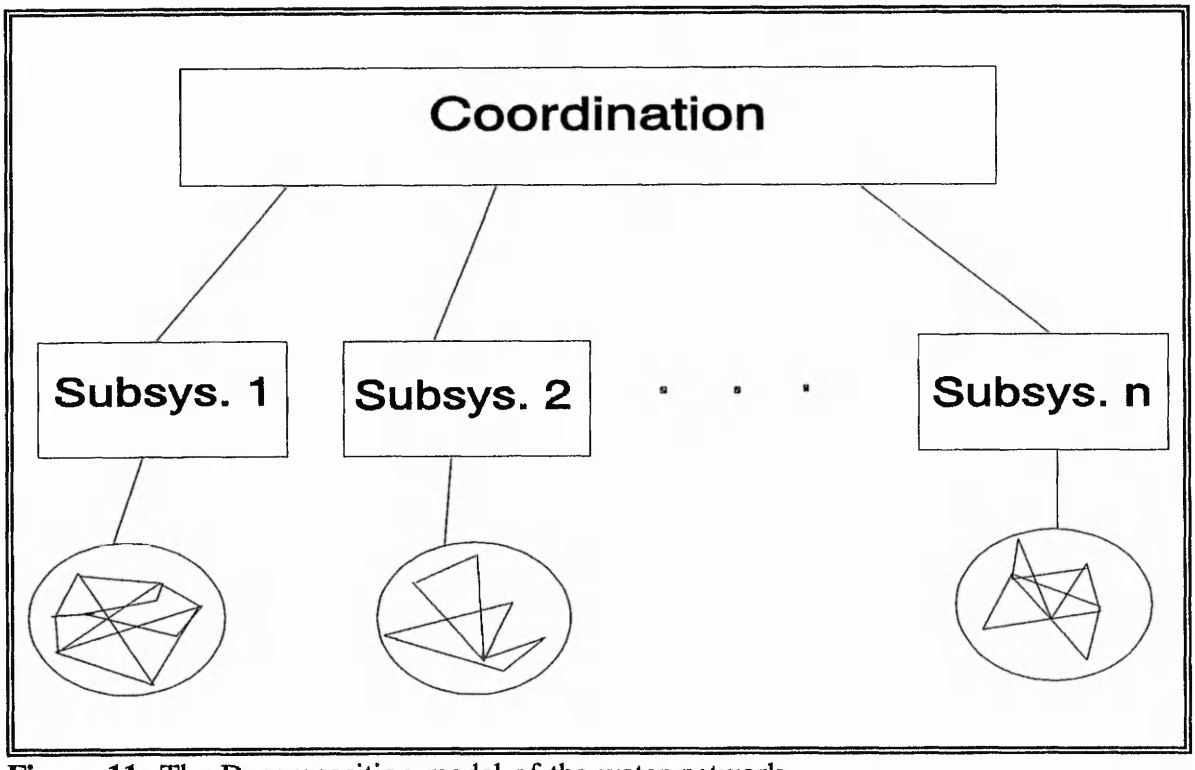

Figure 11: The Decomposition model of the water network.

normally involve the direct solution of a set of matrix equations, including the inversion of the subsystem matrix for each region and the inversion of the system matrix of the intersection network.

The computing required to perform network tearing can be implemented in a hierarchical manner which is illustrated in Fig.12. The subproblems are concerned with solving the subsystem matrices derived from the partitioning of the system matrix into independent blocks using Diakoptics, and the master problem(coordinating routine) combines these solutions with the intersection data to achieve the overall solution. In the following sections of this chapter the special cases of diakoptics of systems with and without common reference point, will be presented.

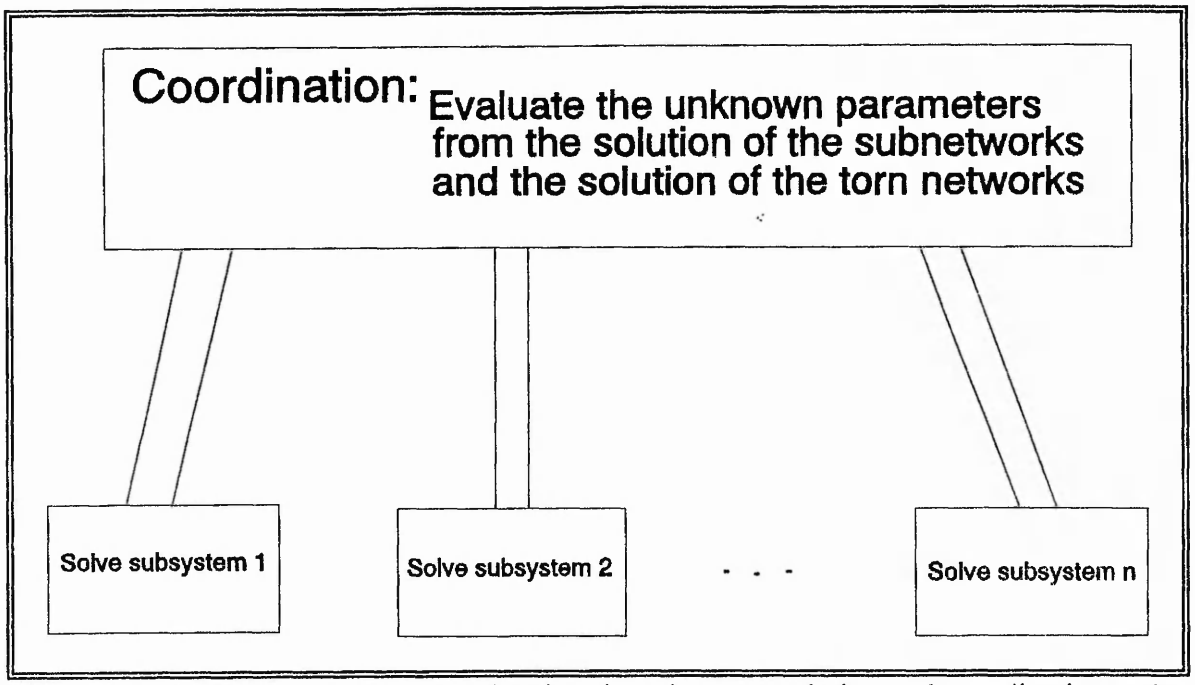

Figure 12: The Decomposed model showing the subsystem solution and coordination tasks.

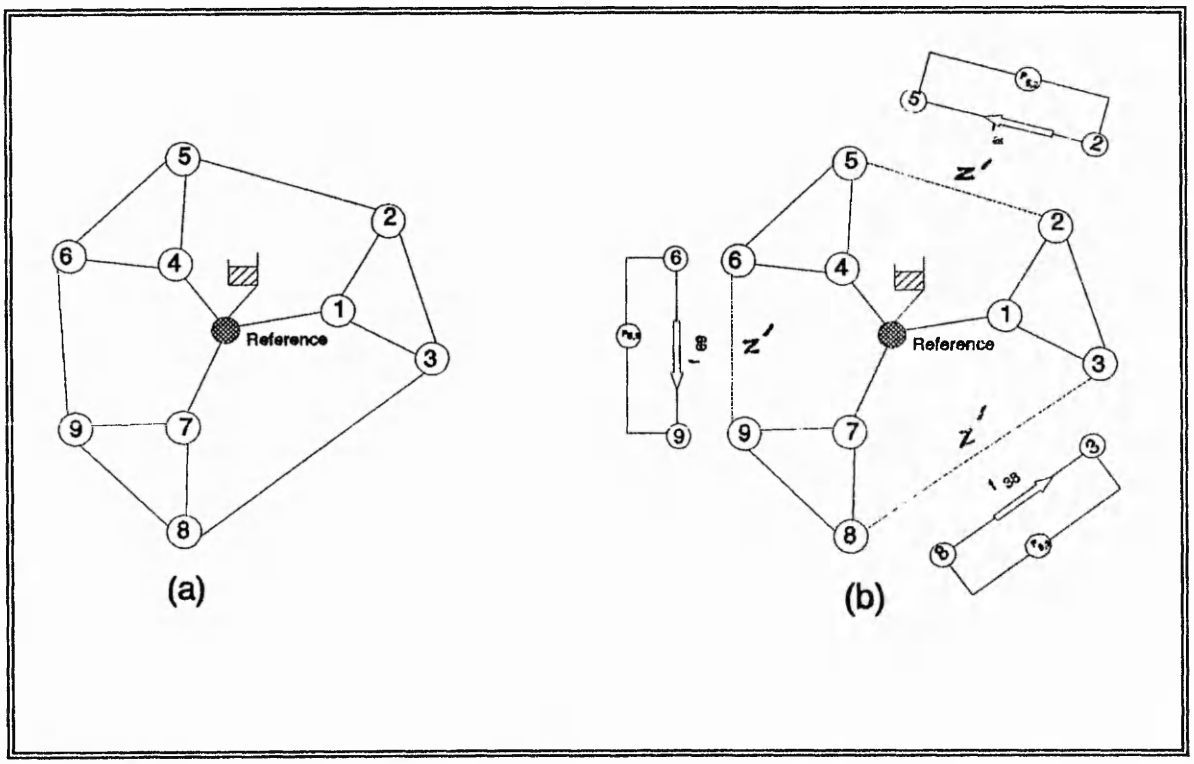

Figure 13: Example network for the common reference point.

3.4.2 Tearing networks with common reference point

With reference to Fig.13, the network is decomposed into three subnetworks by tearing the appropriate pipes(i.e shown as dotted lines in Fig.13(a)). The torn network's system matrix takes the form of a block diagonal matrix and the torn branches form an additional network referred to as the intersection network as shown in Fig13(b). The effects of the tearing can be eliminated by the addition of the compensating flows to the derived subnetworks at the end nodes of the torn branches in the subnetworks. Thus, equation(8) of chapter 1(section 1.3) for such a network shown in Fig.13 would be given by:

$$\Delta x = J^{-1}(Z - g(x^k)) \qquad (51)$$

and the iterative improvement to the solution is obtained as

$$x^{k+1} = x^k + \Delta x \qquad (52)$$

The iterations of (51) and (52) are continued until $|\Delta x|$ is below some pre-set limit ϵ.

While the solution of (51) is theoretically feasible, in practice it implies considerable computational effort due to the need for an inverse of a large Jacobian matrix. Such an inversion has, at best, quadratic numerical complexity. With the network partitioned as in Fig 13(b), the subnetworks can be solved concurrently providing that the effect of the removed interconnecting branches is compensated by adding appropriate flows to the corresponding boundary nodes. Thus the vector Z becomes:

$$Z' = Z + z' \tag{53}$$

where

$$z' = C_{\alpha\psi} f_{\psi}(x) \tag{54}$$

With ψ denoting cut-pipes in the network and $C_{\alpha\psi}$ being a node/cut-pipe incidence matrix defined as follows:

$$C_{\alpha\psi} = \begin{cases} 1, & \text{if the node } \alpha, \text{ is a sending node of the cut-pipe } \psi \\ -1, & \text{if the node } \alpha, \text{ is a receiving node of the cut-pipe } \psi \\ 0, & \text{if the node } \alpha, \text{ is not incident to cut-pipe } \psi. \end{cases}$$

The removal of cut-lines decouples subnetworks so that the Jacobian matrix for the partitioned network assumes block diagonal form and becomes amenable to distributed computation of pressures x' in subnetworks. By analogy to equation (51) can be written as:

$$\Delta x' = J^{-1'}(Z + z' - g(x^k)) \tag{55}$$

Substituting for z' and linearizing cut-line flows around their values assumed for x_0, the correction to the pressures in the subnetworks $\Delta x'$ can be calculated as follows,

$$\Delta x' = J^{-1'}(Z + C_{\alpha\psi} f_{\psi}(x_0) - g(x^k)) + J^{-1'} C_{\alpha\psi} \Delta f_{\psi} \tag{56}$$

and with $Z'_0 = Z + C_{\alpha\psi}.f_{\psi}(x_0)$ (56) becomes:

$$\Delta x' = J^{-1'}(Z'_0 - g(x^k)) + J^{-1'} C_{\alpha\psi} \Delta f_{\psi} \tag{57}$$

Since the corrections to subsystem states $\Delta x'$ are the function of the corrections to cut-line flows Δf_{ψ}, additional equations relating partitioned network pressures and cut-line flows need to be found. These are obtained as pressure balance equations for the removed

networks,(Fig.13(b))

$$h(f_\psi) + P_\psi = 0 \qquad (58)$$

where h(fψ) is a functional relationship between the flow and the pressure drop across the pipe, ψ, and

$$P_\psi = C_{\psi\alpha} X' + M_{\psi\delta} X_\delta \qquad (59)$$

quantifies the pressure drop in terms of the difference of pressures in the end-nodes of the pipe and the difference of reference pressures x_δ in the relevant subnetworks. The cut-pipe/subnetwork incidence matrix, $M_{\psi\delta}$, is defined as follows:

$$M_{\psi\delta} = \begin{cases} 1, & \text{if the flow in the cut-pipe } \psi \text{ is directed into subnetwork } \delta \\ -1, & \text{if the flow in the cut-pipe } \psi \text{ is directed away from subnetwork } \delta \\ 0, & \text{if the cut-pipe } \psi \text{ is not incident to subnetwork } \delta \end{cases}$$

and $C_{\psi\alpha} = C_{\alpha\psi}^T$.

Linearizing (58) and (59) around the x_0, the pressure balance equations for the removed networks can be written as follows:

$$h(f_\psi(x_0)) + \delta h/\delta f_\psi \,.\, \Delta f_\psi + C_{\psi\alpha} x_0 + C_{\psi\alpha} \Delta x' + M_{\psi\delta} \,.\, x_\delta + M_{\psi\delta} \Delta x_\delta = 0 \qquad (60)$$

Assuming that the pressure balance holds for x_0, (60) simplifies to:

$$H \,.\, \Delta f_\psi + C_{\psi\alpha} \Delta x' + M_{\psi\delta} \Delta x_\delta = 0 \qquad (61)$$

where $H = \delta h/\delta f_\psi \,|_{f\psi}(x_0)$.

The corrections to the subnetworks' reference pressures Δx_δ, are related in (61) to the change of flow in cut-pipes Δf_ψ and consequently affect the mass transfers between subnetworks. It is necessary, therefore, to consider equations representing mass balance for the subnetworks:

$$Z_\delta + M_{\delta\psi} f_\psi = 0 \tag{62}$$

where Z_δ is a sum of Z_i in all nodes of subnetwork δ, I_δ, $Z_\delta = \Sigma_{i \epsilon I\delta} Z_i$, and $M_{\delta\psi} = M_{\psi\delta}{}^T$.

Noticing the definition of Z' and the equations (53) -(54), the subnetwork mass balance (62) can be linearized around $\mathbf{x_0}$ to give:

$$Z'_\delta + M_{\delta\psi} \Delta f_\psi = 0 \tag{63}$$

where

$$Z'_\delta = Z_\delta + M_{\delta\psi} f_\psi(x_0) \tag{64}$$

Equation (57), (61) and (63) form the basis for the distributed solution of the nonlinear network system. They can be represented in a matrix form:

$$\begin{bmatrix} J' & -C_{\alpha\psi} & 0 \\ C_{\psi\alpha} & H & M_{\psi\delta} \\ 0 & -M_{\delta\psi} & 0 \end{bmatrix} \begin{bmatrix} \Delta x' \\ \Delta f_\psi \\ \Delta x_\delta \end{bmatrix} = \begin{bmatrix} Z'_0 - g(x^k) \\ 0 \\ Z'_\delta \end{bmatrix} \tag{65}$$

If the block diagonal matrices in J' have much higher ranks than matrices $M_{\psi\delta}$ and $C_{\psi\alpha}$, it is economical to solve concurrently the following auxiliary problems,

$$J' \cdot \Delta x'' = Z'_0 - g(x^k) \tag{66}$$

to obtain the uncoordinated subsystem solutions $\Delta x''$

$$\Delta x'' = J^{-1'} . (Z_0' - g(x^k)) \tag{67}$$

and subsequently coordinating them by taking into account corrections to subnetwork reference pressures and inter-subnetwork flows. Equation (67) allows us to re-formulate (65) and to reduce the coordination problem to the following:

$$\begin{bmatrix} H + C_{\psi\alpha} J^{-1'} C_{\alpha\psi} & M_{\psi\delta} \\ -M_{\delta\psi} & 0 \end{bmatrix} \begin{bmatrix} \Delta f_\psi \\ \Delta x_\delta \end{bmatrix} = \begin{bmatrix} -C_{\psi\alpha} \Delta x'' \\ Z_{\delta'}' \end{bmatrix} \tag{68}$$

On solution to (68) the iterative update to the calculated pressure is found from equations (57) and (67) as:

$$\Delta x' = \Delta x'' + J^{-1'} C_{\alpha\psi} \Delta f_\psi \tag{69}$$

and

$$\Delta x = \Delta x' + K_{\alpha\delta} \Delta x_\delta \tag{70}$$

giving

$$x^{k+1} = x^k + \Delta x \tag{71}$$

which is analogous to (52).

3.4.3 Tearing networks with temporary reference point

In the preceding section, it has been assumed that each component network contains a common reference pressure node to which there are connection from one of more node

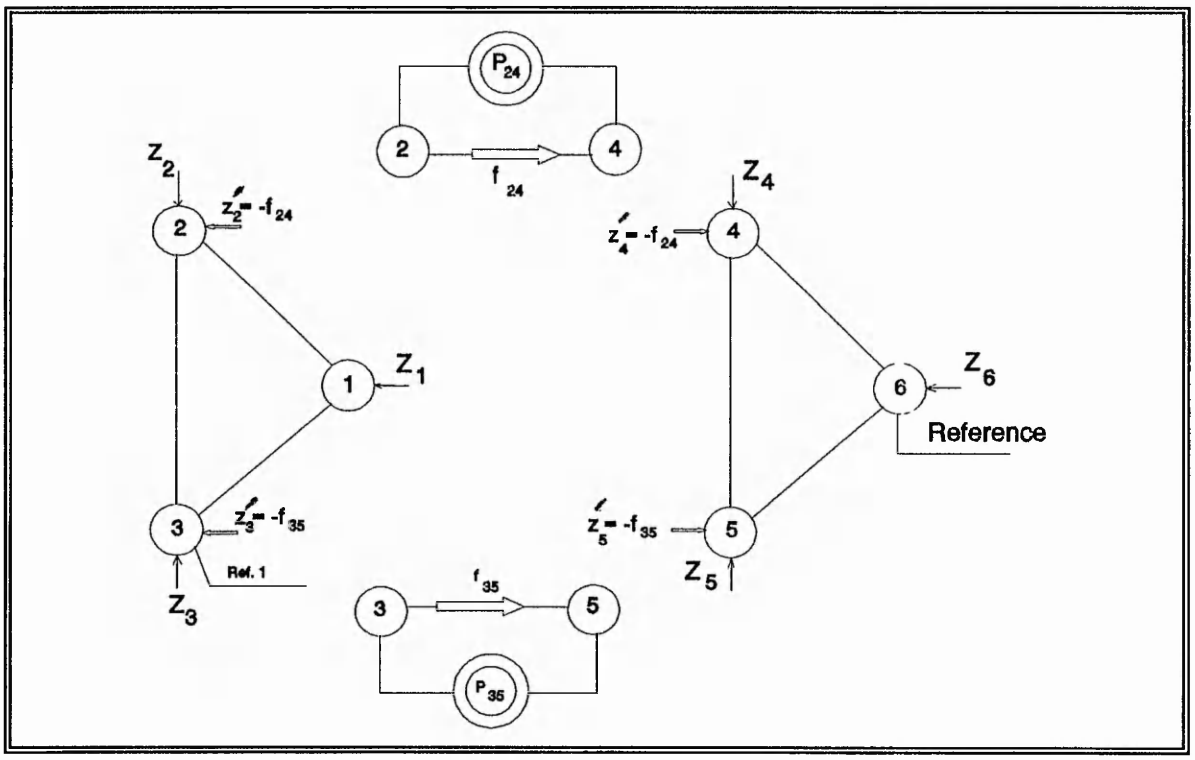

Figure 14: System after partitioning into 2 subsystems together with compensating flows.

in the individual subnetworks. However, it became apparent that in the case of the water distribution system, the reference pressure node would fall in only one of the component networks. Thus, this would mean that only one subnetwork could be solved. An example of such a system is shown in Fig.5. Fig.14 shows the system after partitioning. Like in the previous case, the solution process of the subnetworks requires that the effect of the removed branches be compensated. This is done by adding appropriate flows to the corresponding boundary nodes of the component networks, and is given by:

$$Z' = Z + z'$$

where z' = compensating flows into the end nodes of the cut_branches, in the corresponding subnetwork. By removing the linking components, the system is decoupled, the Jacobian matrix of the partitioned network takes the form of a block

diagonal matrix. This is of course a suitable structure for distributed computation of pressure corrections in subnetworks x'. By partitioning the network, an imbalance is created which is compensated for by the compensating flows z'. Thus the pressure corrections for each subnetwork would be:

$$J'\Delta x' = Z' - g(x^k) \tag{72}$$

where $Z' = Z + z'$

and $z' = C_{\alpha\psi}\, f_\psi$ = flow in the torn network.

Substituting for Z' in (3):

$$J'\Delta x' = Z + C_{\alpha\psi} f_\psi - g(x^k) \tag{73}$$

But f_ψ is nonlinear thus needs linearizing about the equilibrium point as follows:

$$\Delta x' = J^{-1'}\left(Z + C_{\alpha\psi} f_\psi(x_0) + C_{\alpha\psi}\Delta f_\psi - g(x^k)\right) \tag{74}$$

here the rate of change of flow Δf_ψ is an unknown together with $\Delta x'$.

Furthermore, a relationship exists between the pressure drop in each torn pipe and the pressures in the subnetworks measured with respect to each subnetwork reference pressures x . Change of flow in the torn branch would effect the pressures in the subnetwork, measured with respect to their subnetwork reference pressure. Of course that, this relationship needs to be linearized thus resulting in the relationship between the correction to the flow in the torn pipes being equal and opposite to the correction to the

subnetwork pressures.

In addition to two unknown entities $\Delta x'$ and Δf_{ψ} exists another unknown which defines the correction to the pressure node of each subnetwork. Remembering flow balance law, then : " total consumptions in each subnetwork = total input into each subnetwork(or compensation flows)". Once linearized the corrections to the flows in the torn pipes Δf_{ψ}. Therefore, x can be found from the pressure drop in the torn pipes and the $\Delta x'$ can be found from equation(5) by substituting for Δf_{ψ} already found. Once the pressure correction to the subnetwork pressures($\Delta x'$) are obtained they are added to the pressure correction of the local pressures(Δx) in order to find the overall pressure corrections of the system:

$$\Delta x = \Delta x' + \Delta x_{\delta} \qquad (75)$$

and

$$x^{k+1} = x^{k} + \Delta x \qquad (76)$$

3.4.4 The Implementation

Early successes in the computer simulation of water distribution systems **[26,129,11, 72,9,10,8]** prompted the industry to incorporate these techniques in the development of the on_line decision support systems. However, these attempts were largely frustrated by the rapid increase of the computational requirements of the simulation algorithms with the increase of the network size. Even taking the full advantage of the sparsity exploiting techniques, the numerical complexity of the nonlinear network solving algorithms is

quasi_quadratic with respect to network size. Therefore, alternative approaches to the problem had to be thought of, since the use of the existing techniques would result in either the loss of the real_time performance of the on_line decision support systems or uneconomical computational requirements.

The network tearing algorithm presented in this thesis is based on Kron **[80]** network tearing algorithm. 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. The algorithm is basically comprised of Newton_Raphson iterative process with diakoptical calculation of state increments at each iteration. Fig.5 shows a small water distribution network before partitioning, whose Jacobian matrix (Fig.15) has a block diagonal structure augmented by off-diagonal nonzero elements 'x' representing interconnections between (partitioned) subnetworks. This jacobian structure is ideal for parallel processing when the network is partitioned as shown in Fig.14. The Jacobian matrix of the partitioned network assumes the structure as in Fig.15 but without the interconnecting elements.

The nonlinear network tearing algorithm developed in section 3.4.2 can be summarised as follows **[60]**:

Step1: Read-in the system description data.

Step2: Form subsystem data packets and send them to individual solvers.

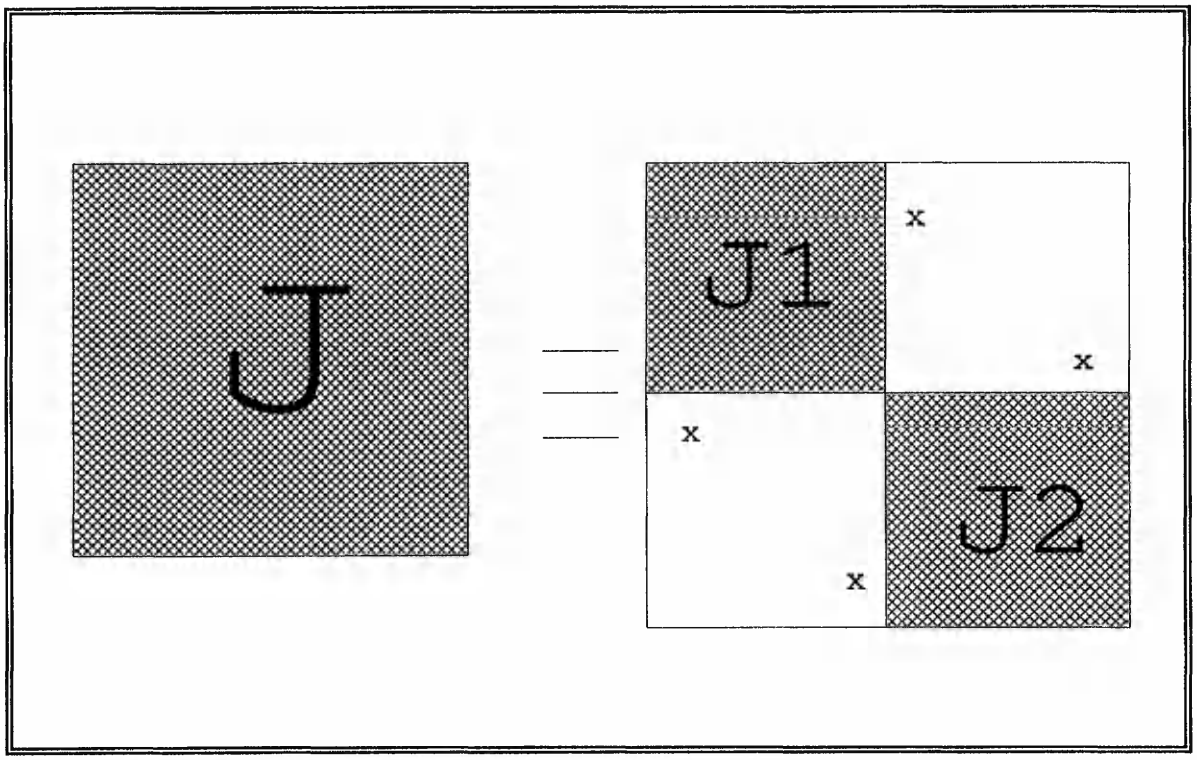

Figure 15: Block structure of the Jacobian matrix.

Step3: Calculate subsystem solutions.

Step4: Coordinate partial solutions.

Step5: If the coordinated corrections from step 4 are less than a given value then STOP otherwise repeat from step 2.

The algorithm was programmed entirely in Parallel Fortran**[149]**. Fig.16 shows the structure of the water system simulation program incorporating nonlinear diakoptics. 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

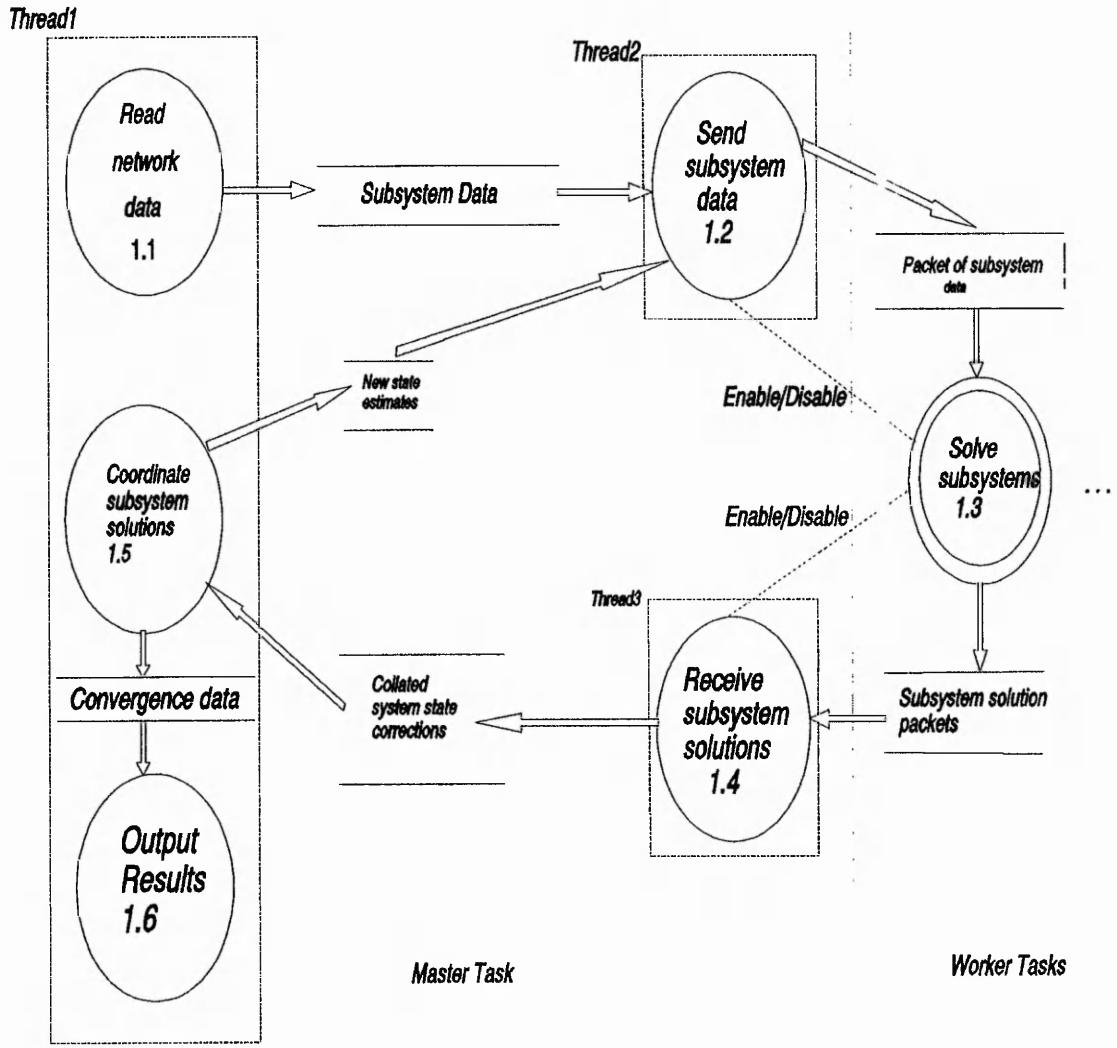

Figure 16: Water distribution simulation program.

convergence criterion the results are output, otherwise the state estimates are transferred to module 1.2 and the cycle of computation is repeated. Since the program was targeted to run on a distributed multiprocessor system, issues such as communication overheads and coordination of currently executing tasks had to be taken into account. To minimize communication overheads modules 1.2,1.4 and 1,5 were programmed as "threads" rather that tasks(this is a facility provided by 3L Fortran**[149]**). In this way, the modules placed on the same processor make use of common memory for accessing data, thus reducing the data transfer time.

Since actual data transfer takes place between modules 1.2_1.3 and 1.3_1.4, the only way of reducing the data transfer time would be to minimize the actual amount of data being transferred. This is achieved for modules 1.2 and 1.3 by actually calculating the Jacobian submatrices within module 1.3 and sending from 1.3 to 1.4 only those columns of the inverse of the Jacobian submatrices which correspond to the end_nodes of the torn branches.

3.4.4.1 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-

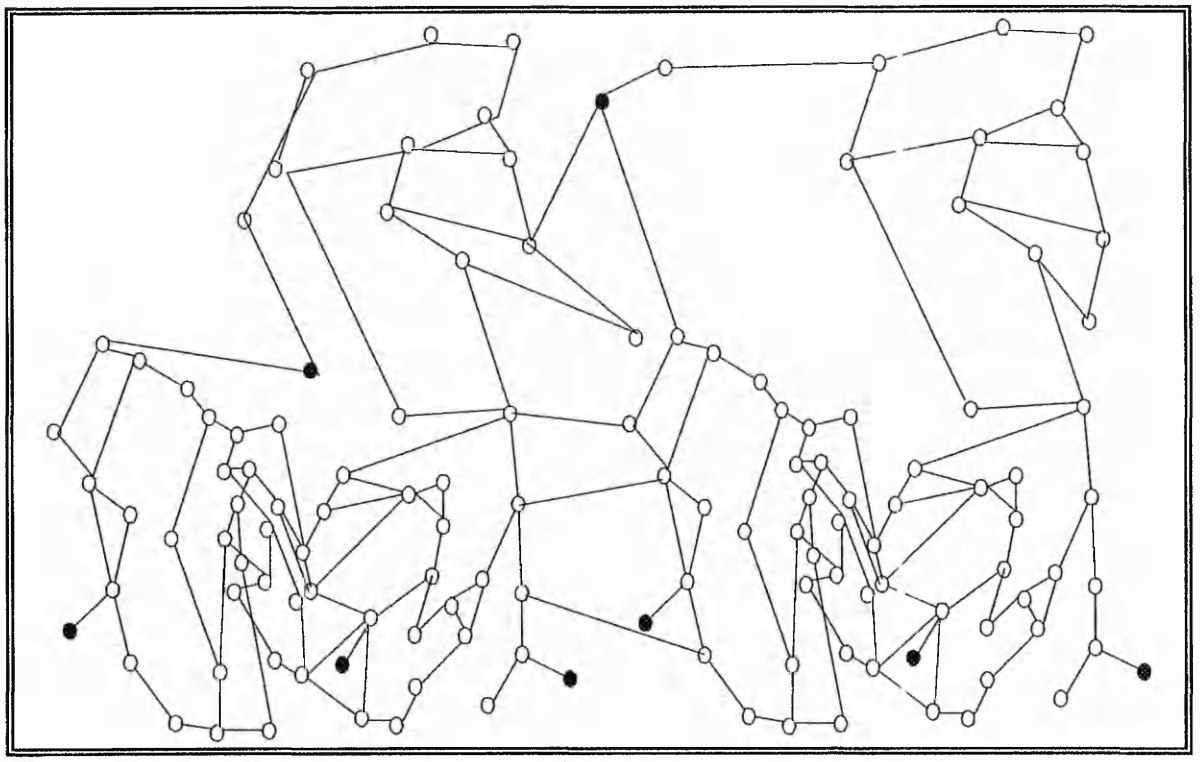

Figure 17: The 130 nodes network example.

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. Fig.18. gives the graphical representation of the performance of pipeline and tree configurations.

As an example, consider a four-processor system(root and 3 workers W1,W2, and W3) and represent the time required for the transfer of results as "R". In the pipeline

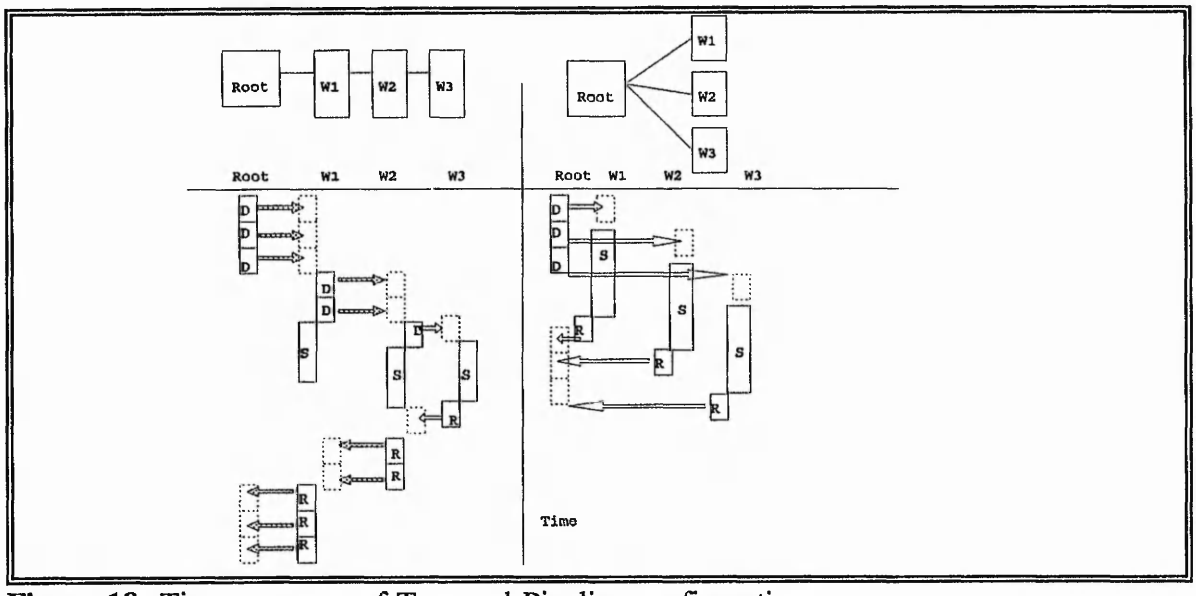

Figure 18: Time response of Tree and Pipeline configurations.

configuration, the root transputer sends 3 data packets of duration D to worker W1. W1 retains one packet of data and passes on the remaining two packets to W2. Similarly, W2 retains one data packet and passes 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 $\mathbf{t_{tot} = 6D+S+6R.}$

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 tree configuration of $\mathbf{t_{tot} = 3D+S+R}$, if D is

greater than R, or t_{tot} =**D+S+3R**, if R is greater then 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. In the context of realistic 65 and 130 node networks, 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 marginal compared to 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 **[16]**, who reported that the tree configuration of the processors executing diakoptics algorithm, does not offer significant advantages over the pipeline configuration. However, compared to **[16]**, the implementation of diakoptics proposed in this thesis requires much lower volume of data transfer.

3.4.4.2 Computational Results.

The algorithm was implemented on a 486 unix workstation hosting a T800 transputer equipped with 4Mb memory and four networked T800 transputers each having 1Mb memory, Figs.19 and 20. The water system simulation program was coded in Parallel_Fortran77. Initially the transputers were configured as a pipeline using the 3L's "flood-fill" configuration software. In flood-fill configuration, tasks are automatically mapped onto the available processors.

However, the result of "flood-fill" configuration suggest that it assigns the worker tasks

to the first two transputers situated immediately after the root transputer in a pipeline configuration (Fig.19). This means that while some of the worker tasks are running on the allocated transputers, the remaining worker tasks are allocated to the root transputer. This phenomenon contradicts the definition of flood-fill configurer as was described before in this chapter.

To verify that flood-fill does not fulfil its role as the automatic configuration tool, a number of programs were developed in order to configure the transputers as a pipeline formation. These programs configured the transputer array as:

i) 1 master task and 1 worker task (i.e. the root processor runs the coordination task and all but one subsystem solution task - Table 1),

ii) 1 master task and 2 worker tasks (i.e. the root processor runs the coordination task and all but two subsystem solution tasks - Table 2),

iii) 1 master task and 3 worker tasks (i.e. the root processor runs the coordination task and all but three subsystem solution tasks - Table 3),

and finally

iv) 1 master task and 4 worker tasks (i.e. the root processor runs the coordination task and all but four subsystem solution tasks - Table 4).

The timing results obtained from configuration(iii) and (iv),shown in Tables 3 and 4, were almost identical to (ii), shown in Table 2. This result confirms that the extra parallelism provided by the tasks in configurations (iii) and (iv) were not utilized.

Furthermore, timing results obtained from configuration (ii) were almost identical to the results obtained while using the "flood_fill" configuration program (Tables 5 and 6).

The performance of the water system simulation program was evaluated on two realistic 65 and 130 nodes networks (see Fig.17).

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 packets had to be appropriately managed by a separate router software since, as was mentioned before, the standard "Flood Configurer" tended to send data packets only to some of the "workers" leaving others idle. The time required to coordinate subsystem solutions,(Fig.21), was to depend almost quadraticly on the coordination problem size, as defined by the combined count of subsystems and the coordination problem size, and the torn branches between them. On other hand, the subsystem solution times were found to depend quasi-linearly(power 1.25) on the number of nodes in subsystems(Fig.22).

These results indicate that while it is advantageous to solve subsystems concurrently one needs to be careful about subdividing the system 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 two 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 minimized (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 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 17sec giving a total of 21secs. This compares very favourably with 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 linearly proportional to the network size, so the data transfer times will continue to be negligible.

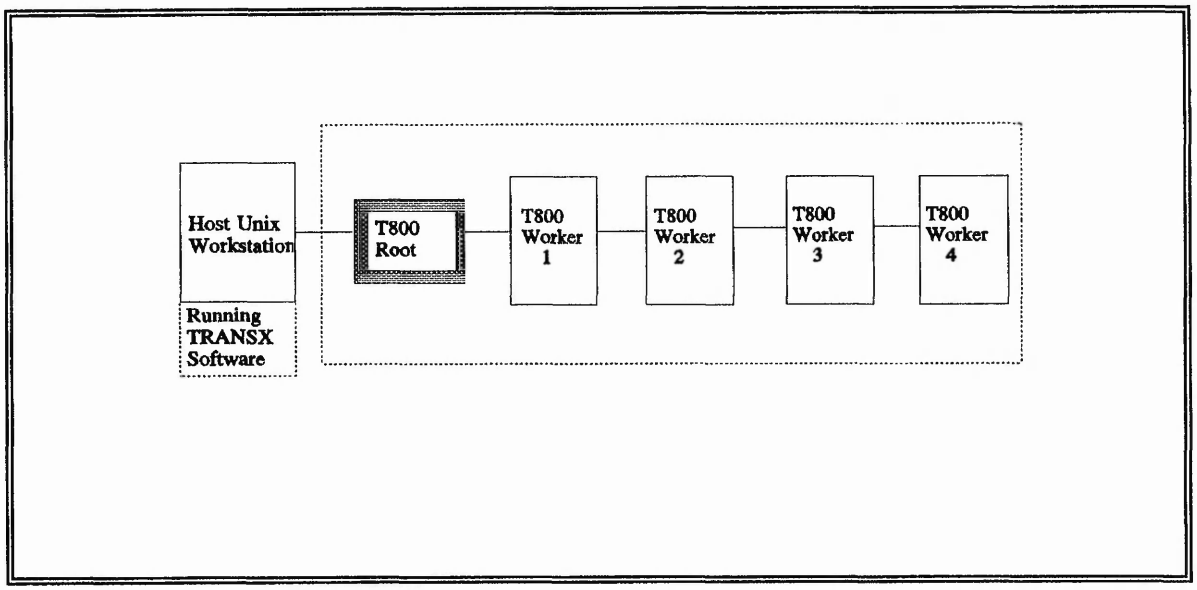

Figure 19: Pipeline configuration.

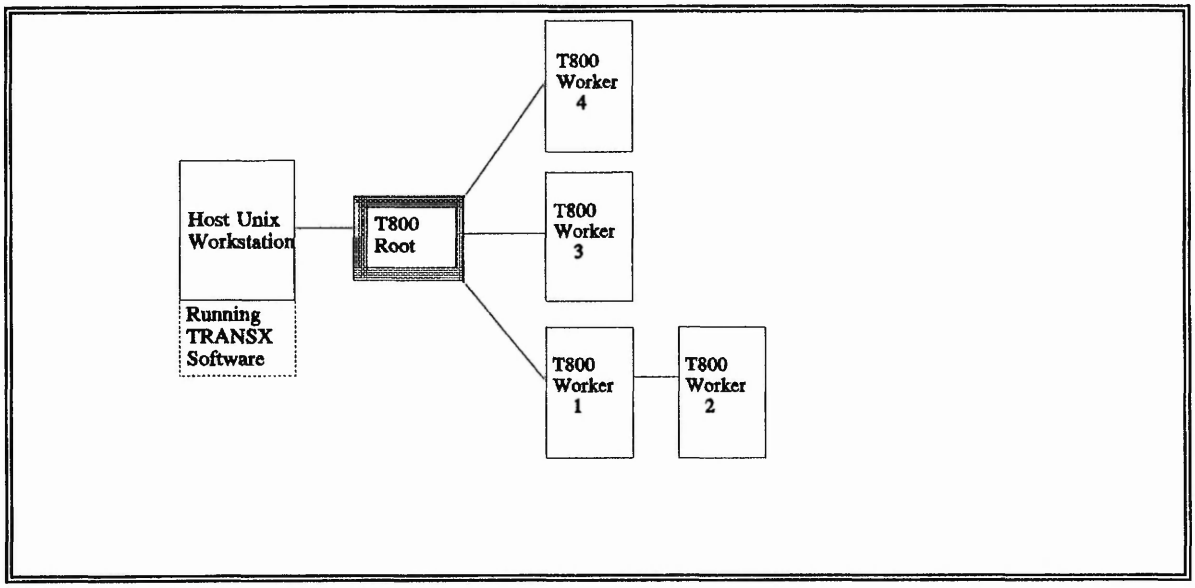

Figure 20: Tree configuration.

Table 1(a): *Pipeline configuration (1 master task and 1 worker task - 65 nodes).*

Number of Subsystems	Number of cut branches	Network size	Subsystem Solution time(Secs)	Coordination time(Secs)
2	5	65	1.928	0.158
3	11		1.760	0.775
4	14		1.516	1.014
5	16		1.448	1.111

Table 1(b): *Pipeline configuration (1 master task and 1 worker task - 130 nodes).*

Number of Subsystems	Number of cut branches	Network size	Subsystem Solution time(Secs)	Coordination time(Secs)
2	4	130	5.851	0.184
3	11		5.351	1.066
4	19		5.2	1.895
6	27		4.695	3.975

Table 2(a): *Pipeline Configuration (1 master task and 2 worker tasks).*

Number of Subsystems	Number of cut branches	Network size	Subsystem Solution time(Secs)	Coordination time(Secs)
2	5	65	1.162	0.158
3	11		1.180	0.775
4	14		0.862	1.014
5	16		0.889	1.11
2	4	130	3.042	0.184
3	11		3.415	1.066
4	19		2.821	1.895
6	27		2.470	3.975

Table 2(b): *Tree configuration (1 master task and 2 worker tasks).*

Number of Subsystems	Number of cut branches	Network size	Subsystem Solution time(Secs)	Coordination time(Secs)
2	5	65	1.155	0.158
3	11		1.181	0.775
4	14		0.857	1.014
5	16		0.897	1.111
2	4	130	3.031	0.183
3	11		3.425	1.066
4	19		2.829	1.895
6	27		2.489	3.975

Table 3(a): *Pipeline configuration (1 master task and 3 worker tasks).*

Number of Subsystems	Number of cut branches	Network size	Subsystem Solution time(Secs)	Coordination time(Secs)
2	5	65	1.162	0.158
3	11		0.912	0.775
4	14		0.755	1.014
5	16		0.745	1.111
2	4	130	3.043	0.183
3	11		2.762	1.066
4	19		2.821	1.895
6	27		2.358	3.975

Table 3(b): *Tree configuration (1 master task and 3 worker tasks).*

Number of Subsystems	Number of cut branches	Network size	Subsystem Solution time(Secs)	Coordination time(Secs)
2	5	65	1.155	0.158
3	11		1.181	0.775
4	14		0.857	1.014
5	16		0.897	1.111
2	4	130	3.031	0.183
3	11		3.425	1.066
4	19		2.829	1.895
6	27		2.489	3.975

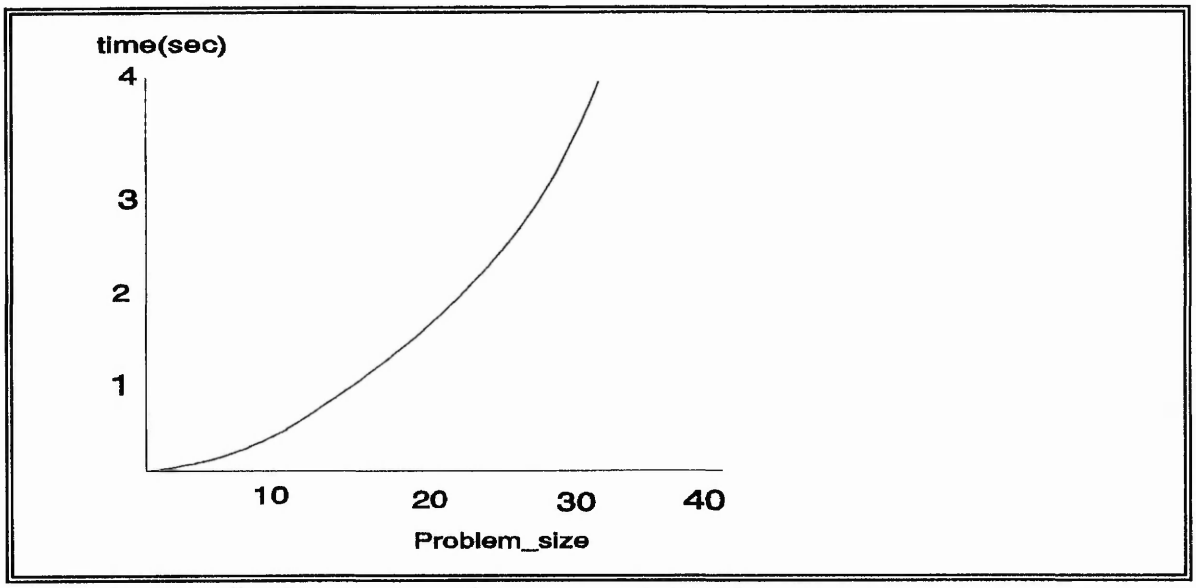

Figure 21: Coordination time / problem_size graph (problem-size represents the total number of subnetworks plus total number of cut-lines).

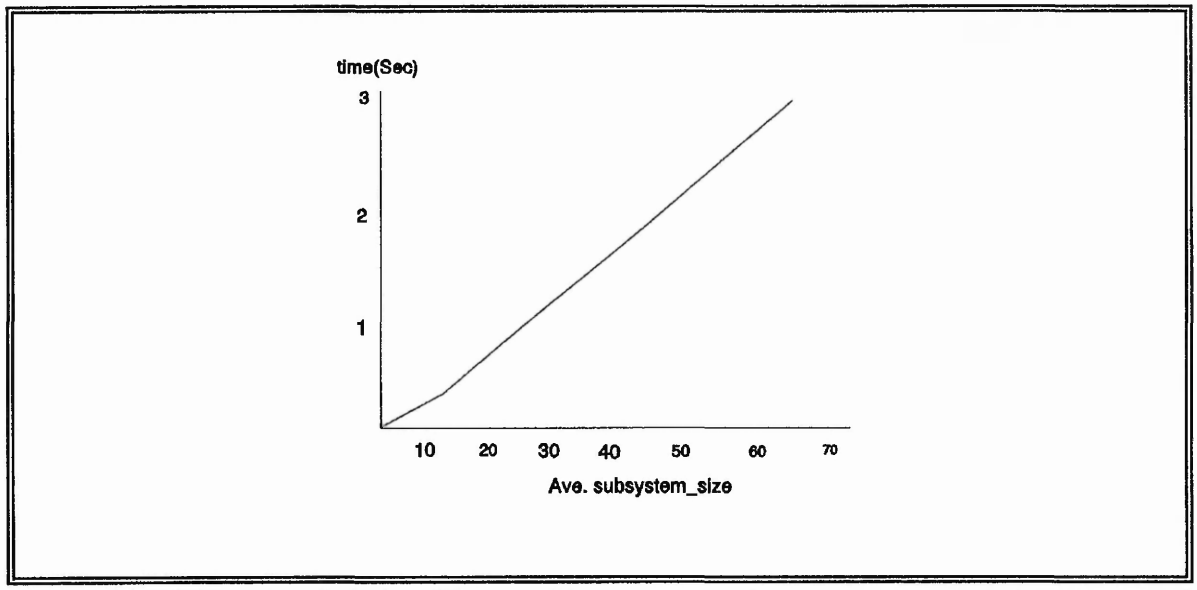

Figure 22: Subsystem Solution Times with respect to Average subsystem-size (i.e. total number of nodes in a subnetwork).

Table 4(a): *Pipeline configuration (1 master task and 4 worker tasks).*

Number of Subsystems	Number of cut branches	Network size	Subsystem Solution time(Secs)	Coordination time(Secs)
2	5	65	1.163	0.158
3	11		0.913	0.775
4	14		0.755	1.014
5	16		0.745	1.111
2	4	130	3.046	0.183
3	11		2.763	1.066
4	19		2.821	1.895
6	27		2.359	3.975

Table 4(b): *Tree configuration (1 master task and 4 worker tasks).*

Number of Subsystems	Number of cut branches	Network size	Subsystem Solution time(Secs)	Coordination time(Secs)
2	5	65	1.147	0.158
3	11		1.188	0.775
4	14		0.857	1.014
5	16		0.913	1.111
2	4	130	3.013	0.183
3	11		3.460	1.066
4	19		2.929	1.895
6	27		2.538	3.975

Table 5: *Pipeline configuration - "Flood_Fill" (1 master task and 4 worker tasks).*

Number of Subsystems	Number of cut branches	Network size	Subsystem Solution time(Secs)	Coordination time(Secs)
2	5	65	1.204	0.15
3	11		0.956	0.735
4	14		0.826	0.961
5	16		0.819	1.053
2	4	130	3.167	0.174
3	11		2.962	1.011
4	19		3.065	1.797
6	27		2.641	3.776

Table 6: *Tree configuration - "Flood_Fill" (1 master task and 4 worker tasks).*

Number of Subsystems	Number of cut branches	Network size	Subsystem Solution time(Secs)	Coordination time(Secs)
2	5	65	1.205	0.150
3	11		1.240	0.735
4	14		0.906	0.961
5	16		0.952	1.052
2	4	130	3.164	0.174
3	11		3.618	1.011
4	19		2.997	1.796
6	27		2.661	3.776

3.5 Conclusion

This chapter introduced the Network tearing or diakoptics. This technique despite being applied to linear electrical systems, was shown to be adaptable for decomposing nonlinear systems such as the water distribution system considered here. The essence of the modification is the application of decomposition technique to the increments of appropriate variables rather than to their absolute values. The experience of implementing and executing on a parallel hardware, seems to indicate that the algorithm is well suited for execution on parallel computing hardware such as a transputer platform. However, when implementing the algorithm care must be taken with regard to the minimization of data transfer. The communication of data and results between the coordination routine and subsystem solver routines is minimized by transferring the individual columns of the subsystem Jacobians related to the cut_pipes only, and the individual column solution of the subsystems. The storage space required by the algorithm is also minimized by a the use of row-column index sparsity storage scheme **[42]**.

CHAPTER 4: Automatic Network Partitioning techniques

4.1 Introduction

The decomposition techniques developed for large scale problems all require the system to be partitioned into subsystems(i.e clusters) such that elements in the same subsystem are "strongly" interconnected, but subsystems themselves are "weakly" interconnected. The overall computational efficiency and the storage requirement of any decomposition method is strongly influenced by the way the system is decomposed. In cases where the system has a simple layout, a fairly good cluster partition can be determined by inspection but for complex systems, an algorithm must be used to systematically partition the associated graph into an optimal arrangement of clusters. Development of such algorithm has proven to be NP_complete (execution time is not bound by a polynomial function of a problem size) optimisation problem**[117]**. No method for exact solution with a computing effort bounded by a power of N has been found for any of these problems. Consequently,"heuristic" methods have been proposed**[67]**. The objective of these methods is to form clusters with graph nodes, while traversing the graph. In every step, the number of nodes adjacent to the cluster is tracked. The order in which the graph is traversed depends on the criteria used to add node to the cluster being formed. In the following sections the two graph partitioning techniques which were studied, will be presented.

4.2 Greedy Cluster formation technique

As was mentioned in the preceding section, since the partitioning problem was shown to

be a non-polynomial bound optimization problem**[117]**, thus heuristic methods have been purposed to solve it. One such method is known as the "Greedy strategy"("greedy" is a common term in the graph literature**[45]** and is attributed to Jackedmonds**[89]**,it means that the algorithm determines the direction for iteration generally by checking for the "cheapest" local condition). The algorithm studied here is based on the concept of "contour tableau" which consists of an array of three columns, as shown in Fig.23. The left most column is call the "iterating set"(IS), the middle column the "adjacent set"(AS), and the right most column the "contour number"(CN).

The entries of the tableau are determined as follows:

Step1: Choose an initial iterating node and store it in IS(1).

Step2: store in AS(1) all nodes that are adjacent to the node in IS(1).

Step3: Place the cardinality of AS(1) in CN(1).

Step4: Let i=1.

Step5: If CN(i)=0, stop!

Step6: Choose the next iterating node, denoted by n_{i+1}, from AS(i) and place it in IS(i+1).

Step7: Update AS(i+1) from AS(i) by deleting the node n_{i+1} and adding the set V representing all nodes adjacent to n_{i+1} that are not already in AS(i).

Step8: CN(i+1)= |AS(i+1)|.

Step9: Let i=i+1, go to step5.

In order to clarify step 7 assume that in AS(i) and AS(i+1) the stored adjacent nodes of the sets of iterated nodes are:

IS	AS	CN
IS(1)	AS(1)	CN(1)
IS(2)	AS(2)	CN(2)
IS(3)	AS(3)	CN(3)
IS(4)	AS(4)	CN(4)

Figure 23: A Contour tableau.

$$\{ \bigcup_{j=1}^{i} IS(j) \}$$

and

$$\{ \bigcup_{j=1}^{i+1} IS(j) \}$$

respectively.

Instead of finding AS(i+1) at each iteration an efficient way of updating AS(i+1) from AS(i) needs to be found. Now sets:

$$\{ IS(i+1) \}$$

and

$$\{ AS(i) - IS(i+1) \}$$

are adjacent to:

$$\{ \bigcup_{j=1}^{i} IS(j) \}$$

Since { AS(i) -IS(i+1) } and V (i.e. set V representing all the nodes adjacent to n_{i+1} that are not already in AS(i) or { $U_{j=1}^{i+1}$ IS(j)}) are adjacent to:

$$\{ \bigcup_{j=1}^{i+1} IS(j) \}.$$

It is now possible therefore, to update AS(i+1) from AS(i) by deleting IS(i+1) and adding V, which is precisely step 7.

To illustrate this methodology an example network is considered. Fig.24 shows a graph with nine nodes. It is clustered into two groups of nodes, $\{n_1,n_2,n_3, n_4\}$ and $\{n_6,n_7,n_8,n_9\}$ which are separated by node n_5. To construct the contour tableau the initial node is selected arbitrarily, say n_1, and stored in IS(1). Since $\{n_2,n_3,n_4,n_5\}$ are the nodes adjacent to n_1, they are stored in AS(1). Consequently, CN(1)=4. Select arbitrarily an iterating node from AS(1), say n_3, and put it in IS(2). Observe that the nodes that are adjacent to $\{n_1,n_3\}$ are $\{n_2,n_4,n_5\}$. Thus they are put in AS(2) and hence CN(2)=3. The algorithm continues in this way until CN(i)=0, resulting in the tableau shown in Fig.25(a).

Moreover, if X denotes the set of nodes of a given graph, then the set of AS nodes always separates X into three subsets; namely,

$$Z(i) = \bigcup_{j=1}^{i} IS(j)$$

AS(i),

and

$$W(i) = X - Z(i) - AS(i)$$

where Z(i) nodes are not adjacent to W(i) nodes. The size of AS(i) (i.e. CN(i)) varies in each step, it is when CN(i) is very small, that Z(i) and W(i) form clusters.

In the contour construction algorithm (described in step1 to step9) there are only two places where choices are made. They are in step1 when choosing the initial iterating node and in step6 when choosing the next iterating node. The methodology adopted in this section for choosing the next iterating node, is that of "greedy" strategy. The greedy algorithm requires that at every iteration only the node in AS(i) that yields minimum $|V|$, be chosen. If a tie is encountered, a node is chosen arbitrarily among the ties.

To illustrate this strategy, examine the graph shown in Fig.24. If n_1 is chosen as the initial iterating node, then applying the greedy strategy to the graph results in the tableau shown in Fig.25(b). Indeed, it achieves the goal of separating the two clusters $\{n_1,n_2,n_3,n_4\}$ and $\{n_6,n_7,n_8,n_9\}$ through the node $\{n_5\}$.

The flow chart of the greedy algorithm is given in Fig.26. Results obtained from the application of the greedy algorism to the water distribution network is shown in Fig.27.

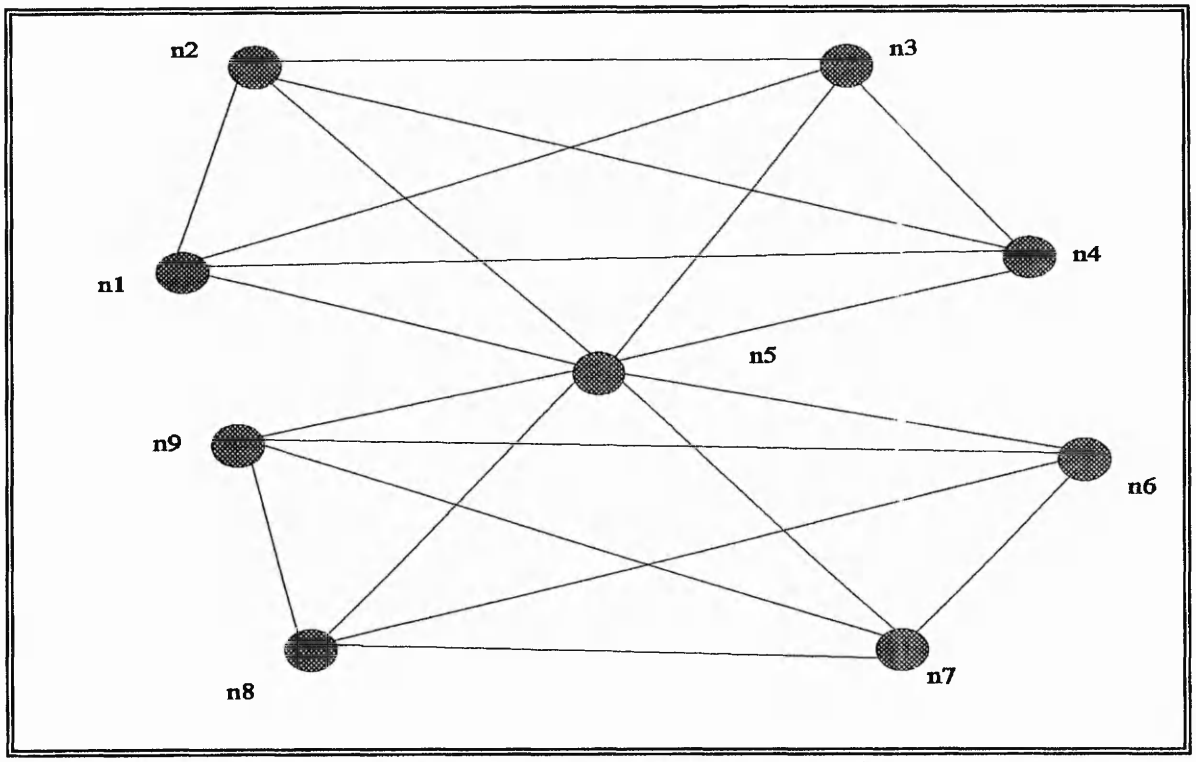

Figure 24: An example network for the contour tableau construction.

IS	AS	CN
n1	n2,n3,n4,n5	4
n3	n2,n4,n5	3
n5	n2,n4,n6,n7,n8,n9	6
n6	n2,n4,n7,n8,n9	5
n2	n4,n7,n8,n9	4
n9	n4,n7,n8	3
n7	n4,n8	2
n4	n8	1
n8	0	0

(a)

IS	AS	CN
n1	n2,n3,n4,n5	4
n2	n3,n4,n5	3
n3	n4,n5	2
n4	n5	1
n5	n6,n7,n8,n9	4
n6	n7,n8,n9	3
n7	n8,n9	2
n8	n9	1
n9	0	0

(b)

Figure 25: Table (a) the results of arbitrarily choosing the next iterating node,
(b) the "greedy" algorithm - selecting from amongst nodes with fewest neighbours which are present in AS(i).

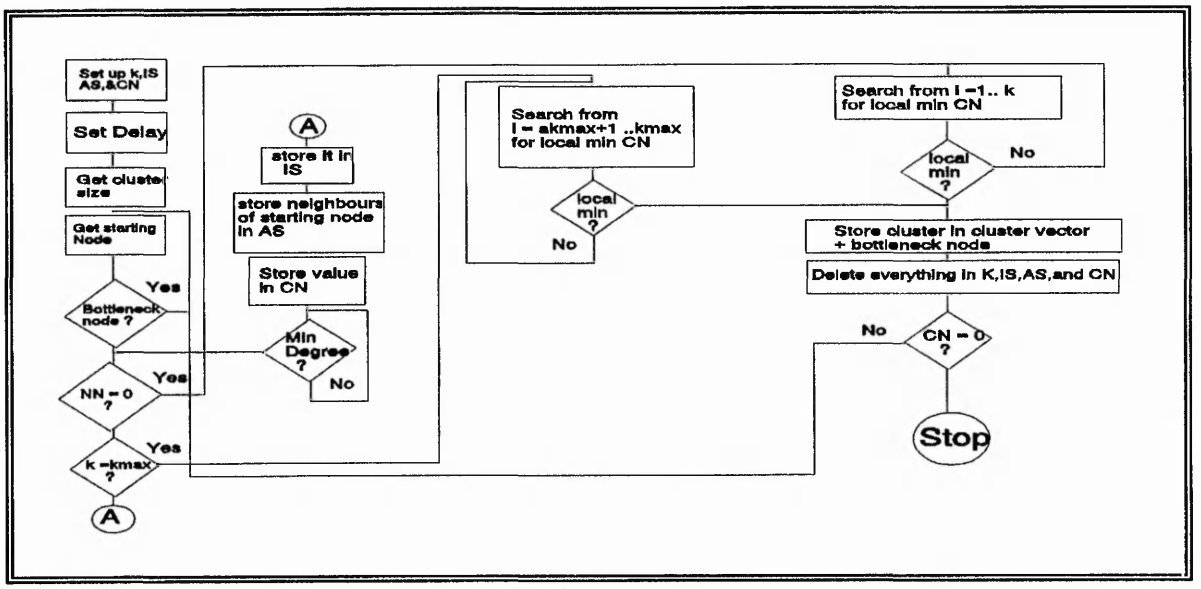

Figure 26: Flow Chart for "greedy" algorithm.

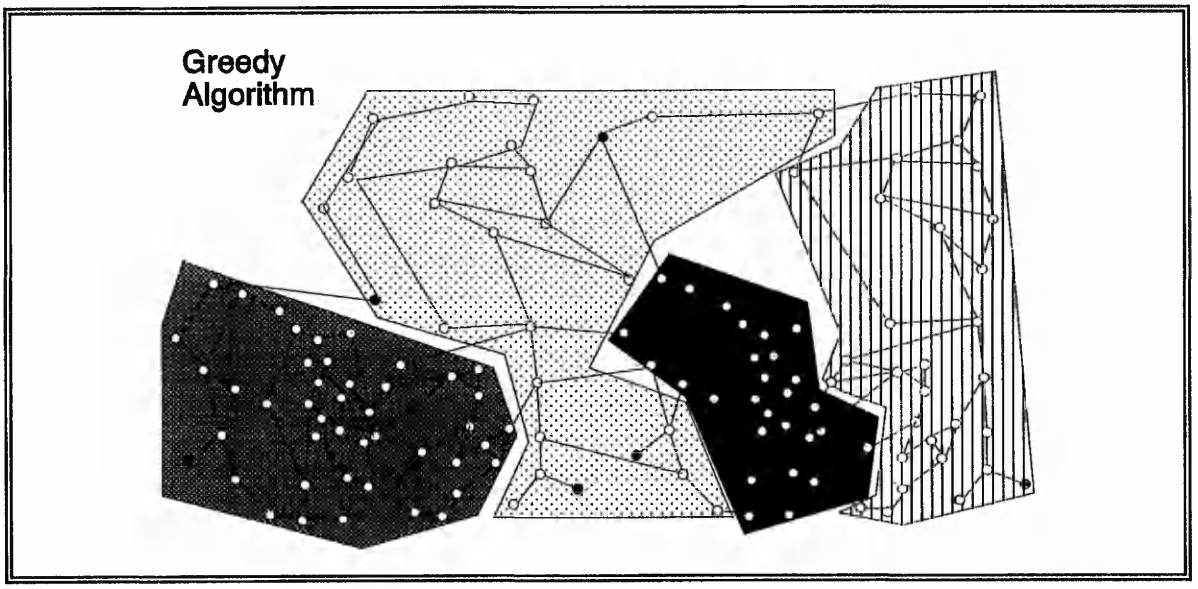

Figure 27: The Greedy algorithm results for a 130 nodes network.

4.3 Simulated Annealing technique

The application of Simulated annealing optimization technique has been reported in fields as wide as electrical engineering , operations research, communications and computer science. This technique basically requires solving large scale combinatorial problems(and the most well known of these problems is known as the travelling salesman problem).

Any combinatorial problem can be defined at any moment in time as a pair(R,C), where R is the finite set of configurations and C is a cost function. The problem now is to find a configuration for which C takes its minimum value. There are two possible approaches for solving a combinatorial optimization problem, firstly is the "optimization algorithm" yielding a globally optimal solution, having extensive amount of computation, secondly is an "approximation algorithm" yielding an approximate solution in an acceptable amount of time. The excessive computation time requirement of the first approach makes it unattractive for computer implementation, thus the remaining approximation method is the only sensible alternative. Approximation algorithms can be divided into two categories: i) algorithms tailored to specific problem, and ii) general algorithms applicable to a wide variety of combinatorial optimization problems.

The simulated annealing algorithm belongs to the second category, it is a general optimization technique for solving combinatorial problems. The simulated annealing algorithm was first introduced by Metropolis et al**[76]** in statistical mechanics. Later on Kirkpatrick et al**[122]** derived the connection between statistical mechanics and

combinatorial optimization. In statistical mechanics, solids are annealed by first raising the temperature to a point where the atoms are randomly arranged and then gradually lowering the temperature and forcing the atoms to arrange themselves into the lowest energy state.

Kirkpatrick et al.**[76]** have proposed an approach to optimisation of NP-hard combinatorial problems based on simulating the process of annealing physical matter. In this model the configuration of a possible solution to the combinatorial problem corresponds to atomic positions and the cost of the configuration corresponds to the internal energy. A low-energy atomic state is reached by very slow cooling, and the equivalent minimum-cost configuration may be achieved in the combinatorial problem by simulating an analogous procedure.

Partitioning of the water network using Simulated Annealing

The objective of the nonlinear network tearing algorithm is to optimize the execution time for the simulation of the water network. In order to achieve this objective, it is essential to derive an optimal partition of the network shown in Fig.17.

An optimal partition in this case (i.e. in the case of nonlinear diakoptics) means dividing the network into a number of subnetworks, while minimizing the number of cut branches required in order to achieve such partitions. Furthermore, the execution time of the nonlinear network tearing algorithm is dependent on its coordination and subsystem solution times. The coordination time reflects mainly the time taken to solve the intersection network (a matrix of interconnecting branches which were cut during the

application of optimal network partitioning) plus the coordination of the subsystem solutions. The subsystem solution time, on other hand, represents the time taken to solve the derived subnetwork. The time is dependent on the maximum number of nodes in any subnetwork (i.e. subnetwork size). Thus the partitioning scheme employed in this case should be chosen in order to achieve a balance between the number of nodes in a subnetwork and the number of branches linking the subnetworks (which are cut in order to obtain the required partition). The estimate of computation time required is given by the following function (Irving et al**[69]**):

$$C = a\Psi^x + b\delta^y \qquad (77)$$

where Ψ = number of cut_branches,

δ = maximum size of the subnetworks,

and α and β are weighting factors. The choice of exponents x and y for ψ and δ respectively, has been determined by the approximate order of solution times required for subsystems solution and full matrix solution of the intersection network in the parallel implementation of the nonlinear network tearing algorithm. As was mentioned in chapter3, the subsystem solution time depends quasi-linearly (power 1.25) on the number of nodes in subsystems **[60]**, while the coordination time depends almost quadraticly on the coordination problem size, as defined by the combined count of subsystems and the number of branches linking the subsystems which were cut during the network partitioning.

In simulated annealing algorithm, searching out the lowest cost configuration is

considered analogous to achieving the lowest energy state of configuration of the physical system by annealing. Here the physical system resembles our example network in Fig.28(a) and the desired optimum cost is plotted against time in Fig.28(b). To ensure that the subnetworks assume an ordered lowest cost state during their formation(or crystallisation) the temperature in the annealing process should be lowered(cooled) very slowly. During this cooling process the system is permitted to move into high as well as low energy state at each temperature, where the probability of it being in a given energy state is determined by the Metropolis criterion**[76]**, which allows the configuration to accept a state which increases its energy by ΔE with probability:

$$\exp\left(\frac{-\Delta E}{T}\right) \qquad \textbf{(78)}$$

where T is the temperature. All configuration changes that do not increase the energy are accepted as an optimization by iterative improvement. However, since the probability of accepting changes to the system state that increases the energy, behaves exponentially, the rate of accepting changes to the energy state will decrease as the temperature T is reduced. The sequence of values chosen for the temperature T during the cooling process is termed the "annealing schedule" and is studied later in this section.

The simulated annealing algorithm applied to the water distribution network therefore, can be described by the following steps:

Step1: Assign each node to an arbitrary subnetwork.

Step2: Set the initial "temperature" T.

Step3: For a number of iterations do:

Step4: For each node in the network:

Step4.1 : Make a trial re_assignment of the node to a randomly selected neighbouring subnetwork(if one exists).

Step4.2 : Evaluate number of cut lines and the size of subnetworks(i.e calculate the cost function).

Step4.3 : If the trial reassignment has reduced the cost, accept the reassignment.

Step4.4 : If the trail reassignment has increased the cost by ΔC,

Step4.4.1 : Generate a random number RN in the range [0,1].

Step4.4.2 : Calculate RP = exp(-ΔC/T).

Step4.4.3 : If RP > RN accept the reassignment, otherwise reject the reassignment.

Step5 : Select next node.

Step6 : Decrease T (T=αT where α=0.99).

Step7 : If the prescribed number of iterations at the current temperature has been reached and no cost reduction is achieved go to exit, otherwise repeat from step3.

Step8 : Write_out results, stop!.

The simulated annealing algorithm is similar to iterative improvement algorithm in that, they start off at a given configuration, a sequence of iteration is generated, each iteration consisting of a possible transition from the current configuration to a configuration selected from the neighbourhood of the current configuration. If this neighbouring configuration has a lower cost, the current configuration is replaced by this neighbour, otherwise another neighbour is selected and compared for its cost value. They terminate

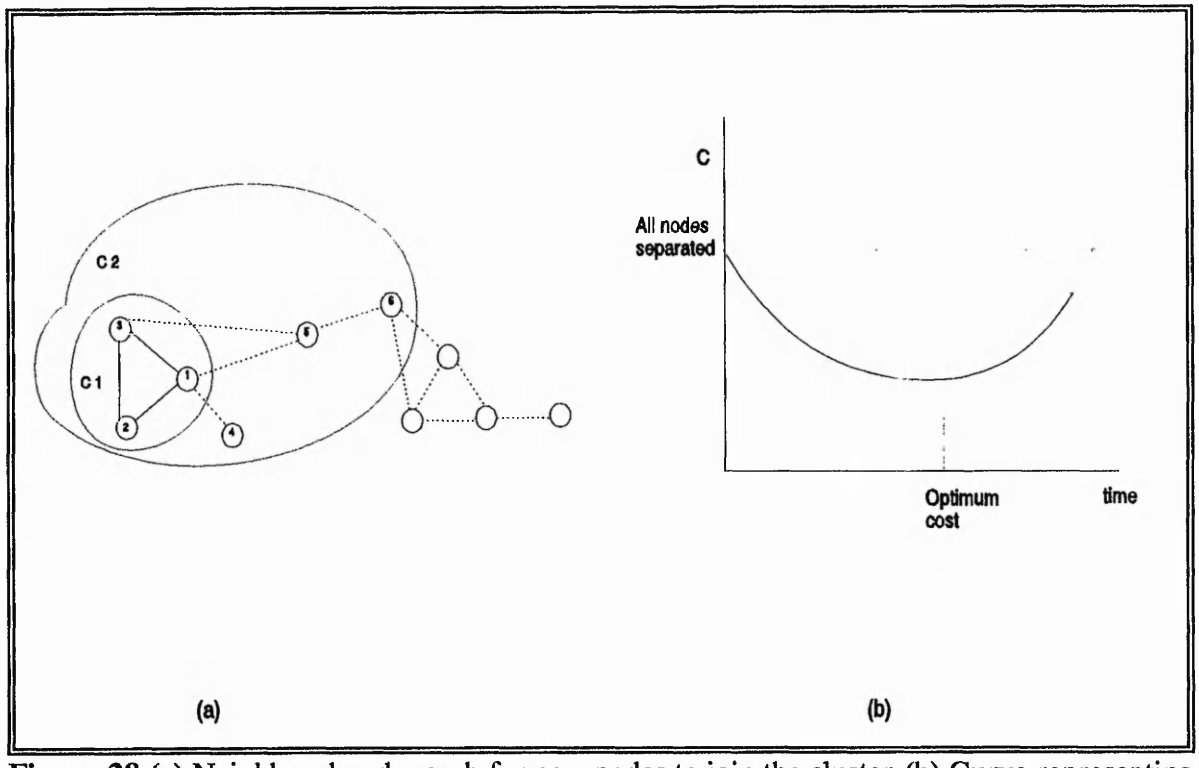

Figure 28 (a) Neighbourhood search for new nodes to join the cluster,(b) Curve representing optimal cost.

when a configuration is obtained whose cost is no worse than any of its neighbours.

Furthermore, step4.4 defines the main difference between the simulated annealing and the iterative improvement method, that is the ability of simulated annealing to "escape" from local optima in an attempt to locate the global optimum. The iterative improvement algorithm is therefore modified to permit the temporary acceptance of trials which increase the value of the cost function. However, since the probability of accepting a trial that increases the cost behaves exponentially (i.e. $\exp(-\Delta C/T)$), the rate of accepting trials that increase the cost will decrease as the temperature T is reduced.

Annealing Schedule

The annealing schedule describes how the temperature T is controlled during minimization of the cost function and hence the probability of accepting changes to configuration that result in an increase to the cost function. The algorithm started with temperature set to a value (e.g T=100,000), where essentially all proposed changes to the configuration are accepted(of course this would depend on the initial configuration, in this case the graph is regarded as unconnected), then reducing the temperature exponentially**[76]**:

$$T_n = \left(\frac{T_1}{T_0}\right)^n T_0 \qquad (83)$$

with the ratio $T_1/T_0 = 0.99$ **[76,122,126]**. At each temperature all possible configurations(i.e all possible neighbouring nodes which could assigned to the subnetwork) are attempted, if they exist. What is of importance here, is the fact that, if one assignment of the neighbouring node to the current subnetwork is rejected, all other neighbouring nodes of the subnetwork under consideration that have not already been examined will be selected with equal probability. Therefore, tables must be set up which keep a record of each node accepted and their neighbours, which effectively become subnetwork's neighbours. The result of the application of simulated annealing to a 130 nodes network is shown in Fig.29. These cluster formations were obtained only at a slow cooling schedule. In contrast however, a rapid cooling schedule resulted in isolated regions being formed which in theory belong to the same cluster.

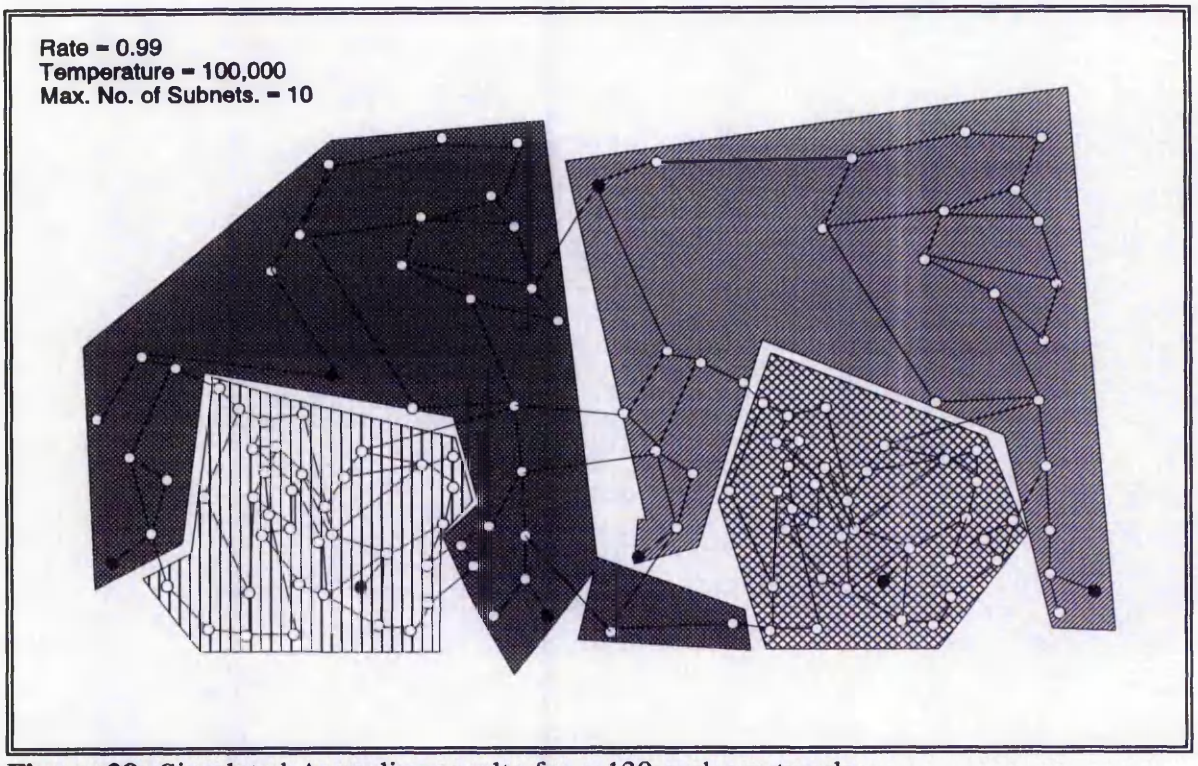

Figure 29: Simulated Annealing results for a 130 nodes network.

The advantages and disadvantages associated with simulated annealing.

The simulated annealing method, being an iterative method, suffers from shortcomings such as: i) iterative improvement ends in a local minimum and there is no information about how far it is from a global minimum, ii) local minimum obtained depends on the initial configuration for which there is no information available, and iii) no bound on computation time.

Moreover, the iterative improvement method does have the advantage of being generally applicable - configurations, a cost function and generation mechanism are usually easy to define. To avoid the aforementioned disadvantages, following measures are recommended in the literature: (i) execution of the algorithm for a large number of initial

configurations, say N at the cost of an increase in computation time; (ii) use of information gained from previous runs of the algorithm to improve the choice of an initial configuration for the next run, (iii) introduction of a more complex generation mechanism(or equivalently, enlargement of the neighbourhoods), in order to be able to "jump out" of the local minima corresponding to the simple generation mechanism. To choose the more complex generation mechanism properly requires detailed knowledge of the problem itself, (iv) acceptance of transitions which corresponds to an increase in the cost function in a limited way(in an iterative improvement algorithm only transitions corresponding to a decrease in cost are accepted).

4.4 Conclusion.

In this section two partitioning schemes were presented. The first scheme has its theoretical basis in a heuristic derivation - the "greedy algorithm". The second scheme is the simulated annealing approach which is theoretically based on the analysis of the combinatorial optimization. The application of ad hoc heuristic methods is strongly problem dependent and thus can not be generalized to tackle broader fields of graph partitioning problems. On the other hand however, the simulated annealing algorithm is a more general approach for graph partitioning problems.

Computational results indicate, however, that the performance of the simulated annealing algorithm is strongly dependent on the chosen cooling schedule, especially as far as the

quality of solution is concerned. In order to apply the algorithm to a particular partitioning problem, a number of items have to be defined: a set of configurations, a generation mechanism for transitions and a cost function. As far as the performance of the algorithm is concerned, it can be concluded that for our implementation the quality of the solution obtained by the simulated annealing algorithm is at least as good as (and sometimes better) than the results obtained from the application of heuristic algorithms(greedy).

CHAPTER 5: Distributed Computing

5.1 Aims and Objectives

A modern water telemetry system is generally comprised of: several out-stations, regional telemetry computer centres and at its heart is a main computing centre. The data gathered by the telemetry out-stations is processed by the software installed at the regional telemetry computer centres. These centres are running a scope of software packages to monitor system data, provide predictions and operational control. In addition, they provide comprehensive information to managers and users, thus bringing about a more effective and near optimum operation of sites. The term used to accurately describe such systems is Supervisory Control and Data Acquisition (SCADA).

The regional telemetry computer centres communicate with each other and other parts of the system via communication links (e.g. Ethernet). This type of arrangement of the processing power within a water distribution system can be viewed as a **Loosely-coupled** computing system (i.e. a system where processing elements communicate solely through message passing). In the case of a distributed physical system, such as a water distribution network, the Loosely-coupled computing system provides flexibility of coordinated subsystems solution. This is particularly important since the periodic expansion of water distribution system can be satisfied by simply increasing the computing power (i.e. simply adding more computers to the network) to deal with network's topological expansion.

The objective here is to develop a suitable environment for the implementation of the nonlinear network tearing algorithm **[60]** in a Loosely-coupled computing network.

5.2 Introduction to Distributed Computing Systems

The progress of computer technology has reached new heights in recent years, this progress is characterised by the introduction of powerful low cost personal computers (PC) and workstations, and the more recently introduced Inmos transputer chip. This

indicates the fact more processing power is available economically for use, thus bringing the processing closer to the hardware it is monitoring or controlling.

Another area of technology which benefitted from this progress is the network communication, thus providing the medium through which processors(PCs, workstations ...etc) can communicate**[32,31]**. Consequently, this gave rise to the construction of large distributed computing systems simply by linking processors together over distances ranging from few meters within a single room to thousands of kilometres across continents. The idea of distributed computing has been utilized to an extent that distributed computing systems comprised of multiple autonomous computers connected through a network are trite in industrial and commercial sectors. The communication between the autonomous processors on the network(i.e. Ethernet,token ring...etc) is usually made possible by the communication program. This layer of software which provides the communication primitives for exchange of data over the network is known as the communication protocol. This layer is situated between the application layer and the physical network layer, thus making the underlying physical network layer **transparent** to the application layer.

5.2.1 Distributed computing system types

The indiscriminate use of the term **distributed system** to describe a wide range of multicomputer and multiprocessor computer systems of differing designs and goals has been the subject of much discussion in the scientific circles. The question is "what type of system actually qualifies as a distributed system?".

From the discussions two lines of thought emerged; firstly were those describing distributed systems as multicomputer and multiprocessor whether in close proximity and sharing resources or remote and communicating over a network by sending messages; and finally were those describing a distributed system to be solely made of geographically distributed computers connected by a communication network(i.e. LAN or WAN). However, the increasing use of distributed systems in all sectors consequently led to their

further development, thus enabling scientists to draw up a definition in order to narrow the field down to**[6]**: "A distributed computing system consists of multiple autonomous processors that do not share primary memory , but cooperate by sending messages over a communication network". Each processor in such a system executes its own instruction stream(s) and uses its own local data, both stored in its local memory. Occasionally processors may need to exchange data; they do so by sending messages to each other over the network.

The most important characteristic of a distributed system is "transparency". As far as the application programmer is concerned, transparency eliminates the need to program the communication operations explicitly, providing instead a range of interfaces to remote services enabling application programs to access files, devices and system resources wherever they are located in the network.

Distributed systems can be characterised by their interconnection networks. The network determines the speed and reliability of interprocessor communication, and spatial distribution of the processors. The following sections describe different types of distributed systems.

5.2.1.1 Tightly-coupled distributed systems

A tightly-coupled distributed system consists of a multiprocessor system where the processors communicate through shared variables, where a shared variable may be read from or written to by an arbitrary number of processors.

Design of concurrent processing systems requires deep analysis. Since speed of component processes are assumed to be nonzero and finite, but otherwise arbitrary, it is necessary to analyze all possible execution sequences, however unlikely some of them may be , to guarantee the absence of "race conditions"(i.e. where two processes are accessing a variable simultaneously). Special protocols for mutual exclusion are often required for a process to access shared-variables in an exclusive manner.

For tightly-coupled systems shared-variables provide clear and better solutions to the mutual exclusion problem. Broadcasting a message can often be implemented by storing the message in a variable that can be read by every process. Examples of tightly-coupled systems are**[6]**: the cosmic cube**[125]**, hypercube**[111]**, and Transputer networks**[87]**. Fig.30 show an example of a tightly-coupled computing system.

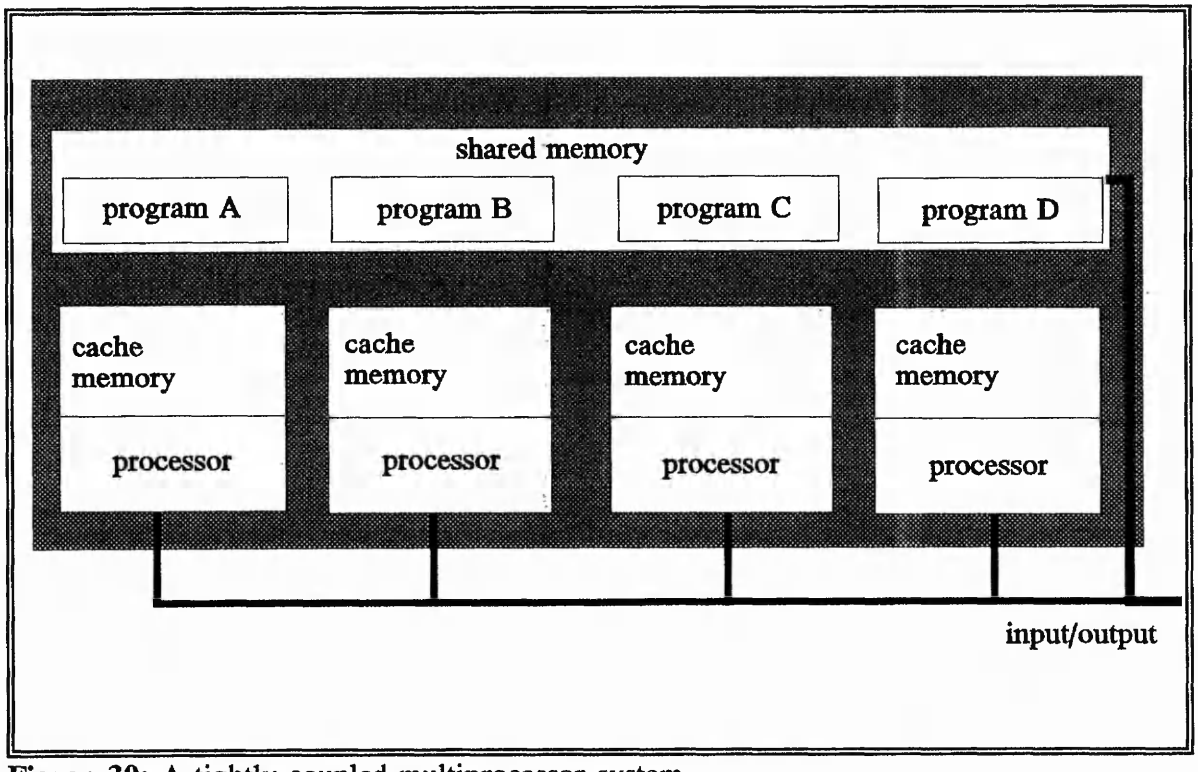

Figure 30: A tightly-coupled multiprocessor system.

5.2.1.2 Loosely-coupled systems

A system of processors in which the interactions are solely through messages, is called a Loosely-coupled distributed system. Such systems are attractive from the programming view point since, they are designed by decomposing a specification into its separable components, each component could then be implemented by a process running on a different processor. A further advantage of using Loosely-coupled system in contrast to a tightly-coupled systems is that in a loosely-coupled system the required variables are sent in message packets between processes, therefore a process can commence its computation immediately after receiving the message packets. In a tightly-coupled system, the mutual

exclusion between processes is achieved by sending synchronisation messages (e.g. ADA rendezvous, or distributed wait signal). This increases the data access time for each process, thus increasing the overall execution time. An other problem associated with tightly-coupled systems, is the possibility of race conditions existing between processors accessing shared variables(as was mentioned before). The only solution is to encapsulate such shared variables in a procedure which incorporates mutual exclusion access to its resources. In contrast,the executable section of each process, in a loosely-coupled systems,is performed in a serial manner thus enforcing mutual exclusion access to shared variables with each distributed process. Fig.31 shows an example of a loosely-coupled system.

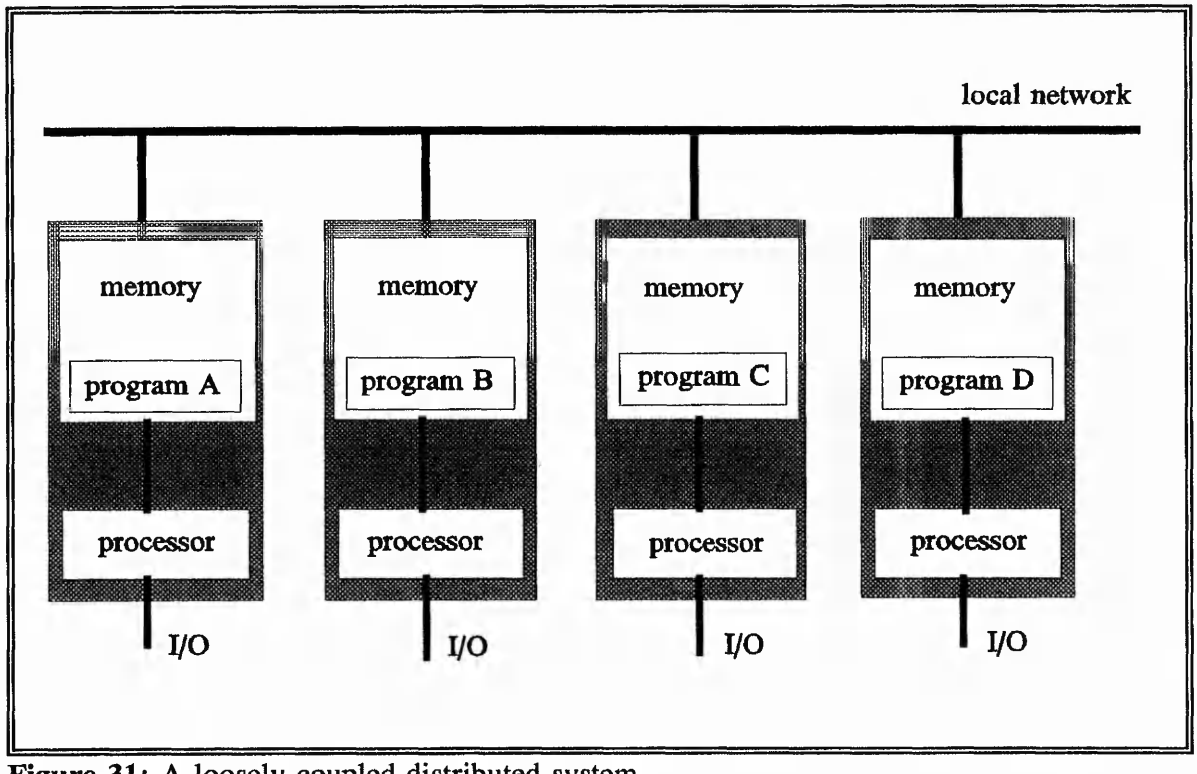

Figure 31: A loosely-coupled distributed system.

5.2.2 Distributed computing architecture of a water distribution system

The basic infrastructure of a water distribution system in use today consists of a large

number of out stations monitoring and collecting system data, and a SCADA system which utilizes the data to provide the overall water system operational control. The SCADA systems are in operation autonomously at large water treatment works and reclamation centres. Such systems enable operators to monitor pressures and flow rates throughout the distribution network and operate various elements(i.e. pumps and valves) from a central location. The objective of controlling the water distribution system is to achieve hydraulic performance required by consumers in an economically efficient manner. For example, the purpose of optimal control for pump-cost minimization is to provide the operator with the least-cost operation policy for all the pump stations in the water distribution system. Therefore, the operation policy for a pump station is simply a set of rules or a schedule that indicates when a particular pump or a group of pumps should be turned on or off over a specified period of time. The optimal policy should result in the lowest total operating cost for a given set of boundary conditions and system constraints.

In general, the optimal control is directly integrated with an associated SCADA system; alternatively the control system may be developed as an independent annex of the overall operating environment. Thus , the control system can be viewed as an ensemble of optimization and network analysis and simulation routines that run on a federation of dedicated personal computers and workstations connected by a network(i.e. Ethernet, token ring ...etc). In fact the water system simulation program described in this report is primarily destined to run in such a computing environment. As was described in the preceding sections, such clusters of computing power resemble a loosely-coupled distributed computing system.

5.2.2.1 Computer Network structure

The communication between distributed computer systems is established by well-defined protocols. Since these protocols can be complex, they are designed in layers to make their implementation more manageable. For each layer there are protocols which constitute a framework for the communication at that layer. However, because of the independent

development of the layers, they tend to vary across networks. To harmonise these differences the ISO (International Standard Organisation) Open System Interconnection model (OSI) **[131]** were introduced(Fig.32). The complete description of different layers in this model can be found in **[130]**. In the ISO model of the network communication system, the lowest layer providing user-process to user-process communication is the transport layer. Thus, whatever the communication protocol for process interaction, it would be built on top of the transport layer of the host communication system. The protocol layer would prevent the application programmer from explicitly programming communication operations, providing instead a range of primitives for connecting to remote services required by the application program. Therefore, making the communication protocol layer "transparent" to its users.

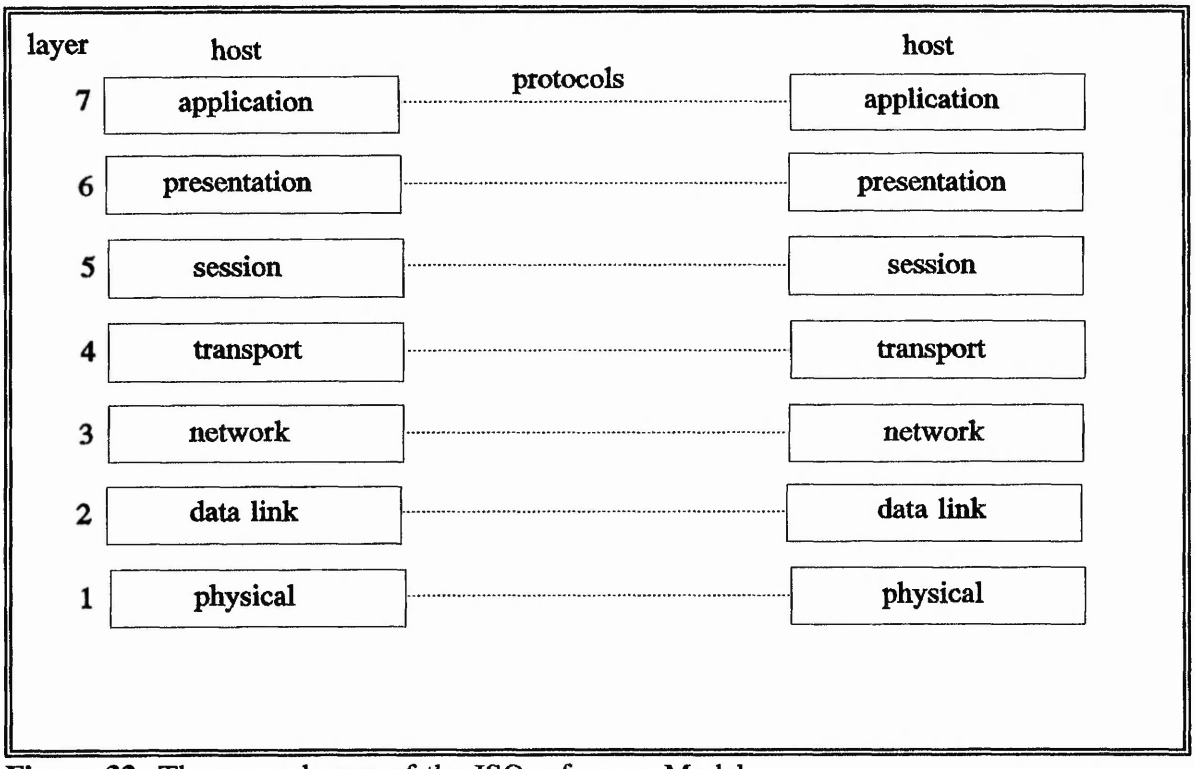

Figure 32: The seven layers of the ISO reference Model.

To establish communication between user-processes communication primitives "SEND" and "RECEIVE" are perfectly adequate, but there are many combinations of the

primitives depending on their connection pattern, for example, many-to-one and one-to-one interprocess communication, and whether or not they are synchronous(i.e. blocking). The most common combination in most languages is a synchronous RECEIVE primitive and a synchronous SEND primitive. In the absence of synchronous RECEIVE and SEND, periodic monitoring of the receiving port is required.

5.2.2.2 Protocols for interprocess communication

The interprocess communication protocol suite provided by UNIX 4.3BSD system are: TCP/IP protocol suite(the Internet family), Xerox Networking systems(Xerox NS), the OSI protocols and Unix-to-Unix copy(UUCP). These protocols would be described in the following sections. However, since in this report the emphasis is on the usage of TCP/IP protocols, it was not deemed necessary to describe the other protocols in any great depth. Thus, the inner workings of these protocols were glossed over in favour of a more general description of their important aspects.

The Internet Protocols

In order to satisfy the networking needs of the US department of defense (DoD), the Advance Research Project Agency (ARPA) of DoD sponsored the development of the ARPANET. Consequently, in 1980s a new family of protocols referred to as the TCP/IP protocol suite was specified as the standard for the ARPANET. The TCP/IP has been implemented on everything from personal computers to the largest supercomputer. It is used for both Local Area Networks(LANs) and Wide Area Networks(WANs). The reason for the increased use of the TCP/IP protocols is their inclusion in the BSD UNIX system. The existence of this protocol along side of BSD Unix in workstations allowed many organizations and university departments to establish their own LANs.

Although the protocol family is referred to as TCP/IP, the family has more members than TCP and IP. Fig.33 shows the relationship of the protocols in the protocol suite along with their approximate mapping into the OSI model (approximate because the TCP/IP was developed before the OSI model became available).

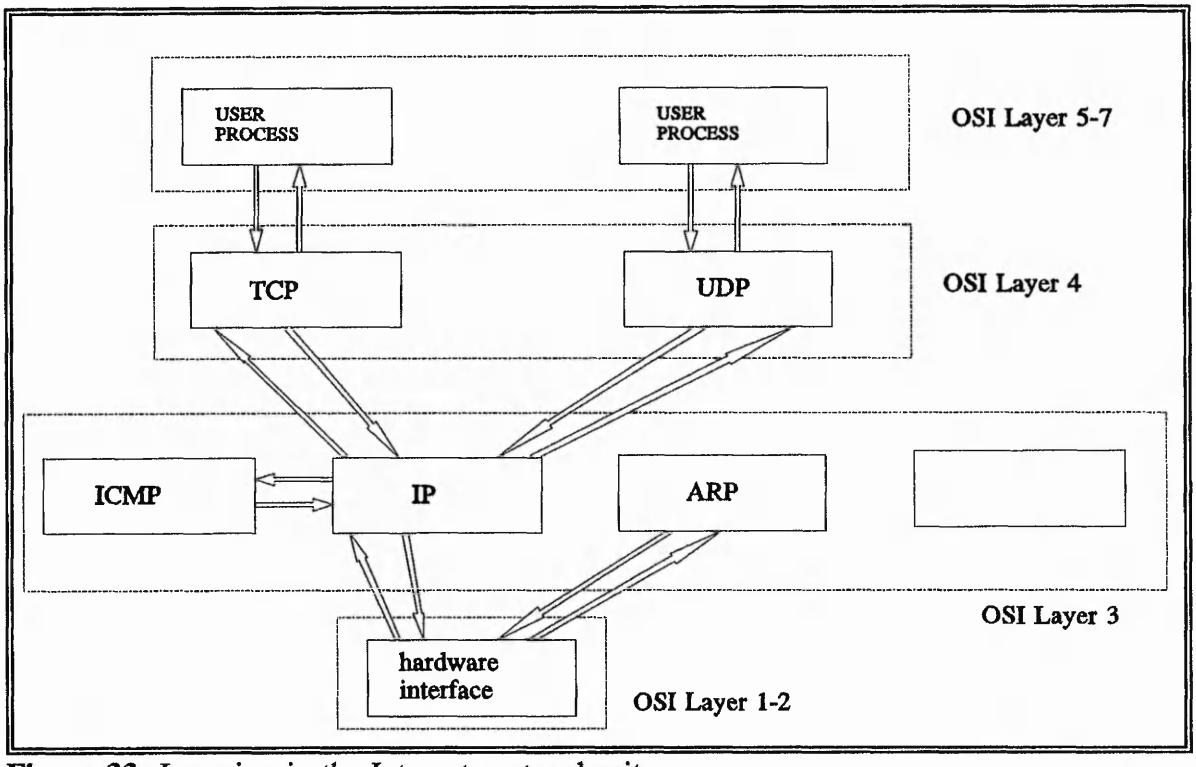

Figure 33: Layering in the Internet protocol suite.

The TCP(Transmission Control Protocol) is a connection-oriented protocol that provides a full-duplex, byte stream communication link for the user process. The UDP(User Datagram Protocol), on the other hand, is a connectionless protocol for user processes. The data is transmitted as datagram packets each having in its header the destination address of the receiver. The Internet Protocol(IP) is yet another protocol that provides a packet delivery service for the TCP and UDP protocols. User processes are not involved with the IP layer.Finally there are those protocols whose main role is to map an Internet address into a hardware address - they are Address Resolution Protocol(ARP) and Reverse Address Resolution (RARP). Some networks may require them but the majority do not.

User processes interact with TCP/IP protocols by sending and receiving either TCP data or UDP data. These two protocols are referred to as TCP/IP or UDP/IP to indicate that

both use the IP layer. The TCP module provides the necessary logic for establishing and terminating connection between processes, the sequencing of data that might received out of order, the end-to-end reliability and the end-to-end flow control. UDP, on the other hand, provides only two features that are not provided by the IP protocol; port numbers and an optional checksum to verify the contents of the UDP datagram.

Port Numbers

It is possible for more than one process at a time to be using either TCP or UDP protocols. This requires a mechanism for identifying the data associated with each user process. This is the reason why both the TCP and the UDP protocols use a 16-bit integer number as their "port numbers". Thus if a client want to contact a server, the 32-bit Internet address of the host on which the server resides would be combined with its port number to form a unique network wide address. The UDP and TCP headers both contain the source port number and the destination port numbers. The TCP ports are independent of the UDP ports, since the IP header specifies the protocols.

The inclusion of the control information by the different protocol modules is known as "encapsulation". The combination of information from different sources using identifiers such as port numbers, protocol types, and Internet addresses is called "multiplexing".

Both TCP and UDP use the IP layer, if however the size of the packet, passed down from these layers, to the IP layer is greater than the Maximum Transmission Unit(MTU) of the network access layer, the packet is fragmented before it is passed to the network access layer. Upon receiving the fragmented packet, the receiving IP layer has to reassemble the fragments into a single datagram before it is passed to either UDP or the TCP layer. Whether fragmentation takes place or not, the size of the data packet exchanged by the two UDP(TCP) layers remains the same.

The UDP datagram created by a user process, is sent by the IP layer as soon as it arrives in the IP layer. Thus the concept of buffering does not apply to UDP. TCP layer, on the

other hand, provides this facility.

The maximum size of an IP datagram is 65,536 bytes. However, as soon as the size of an IP datagram exceeds the size of the underlying network's MTU, fragmentation occurs. Since UDP packets are transmitted using the IP layer, if the result of adding the UDP header and the IP header causes the datagram to exceed the network's MTU, again fragmentation occurs (as mentioned previously). This means that sending a 2048 byte UDP packets on an Ethernet network guarantees fragmentation. TCP functions differently since it breaks up the data into segments. The segment size used by TCP layer is agreed on between the two ends when a connection is established.

Xerox Network Systems

Xerox Network Systems(XNS) is the network architecture developed by Xerox corporation in the late 1970s for integrating their office products and computer systems. XNS is an "open" system(i.e. open-ended, easily expandable), and Xerox has published and made available the protocols used by XNS. Most 4.3BSD systems provide support for the XNS protocol suite. XNS is similar in structure to the TCP/IP protocol suite. It provides several different protocols, some of those are: Echo protocol(ECHO), Routing Information Protocol(RIP), Packet Exchange Protocol(PEX), Sequenced Packet Protocol(SPP), Error Protocol(ERROR) and finally Internet Datagram Protocol(IDP). The arrangement of the layers in the XNS protocol suite, and their approximate mapping onto the OSI model is shown in Fig.34.

OSI Protocols

OSI protocols have become popular lately and many organisations(e.g. US government) have stated their intentions to move towards network based on OSI standards. Unfortunately networks based on OSI protocols are still in their infancy. However, a nonproprietary implementation of many of the OSI protocols is available as the ISODE

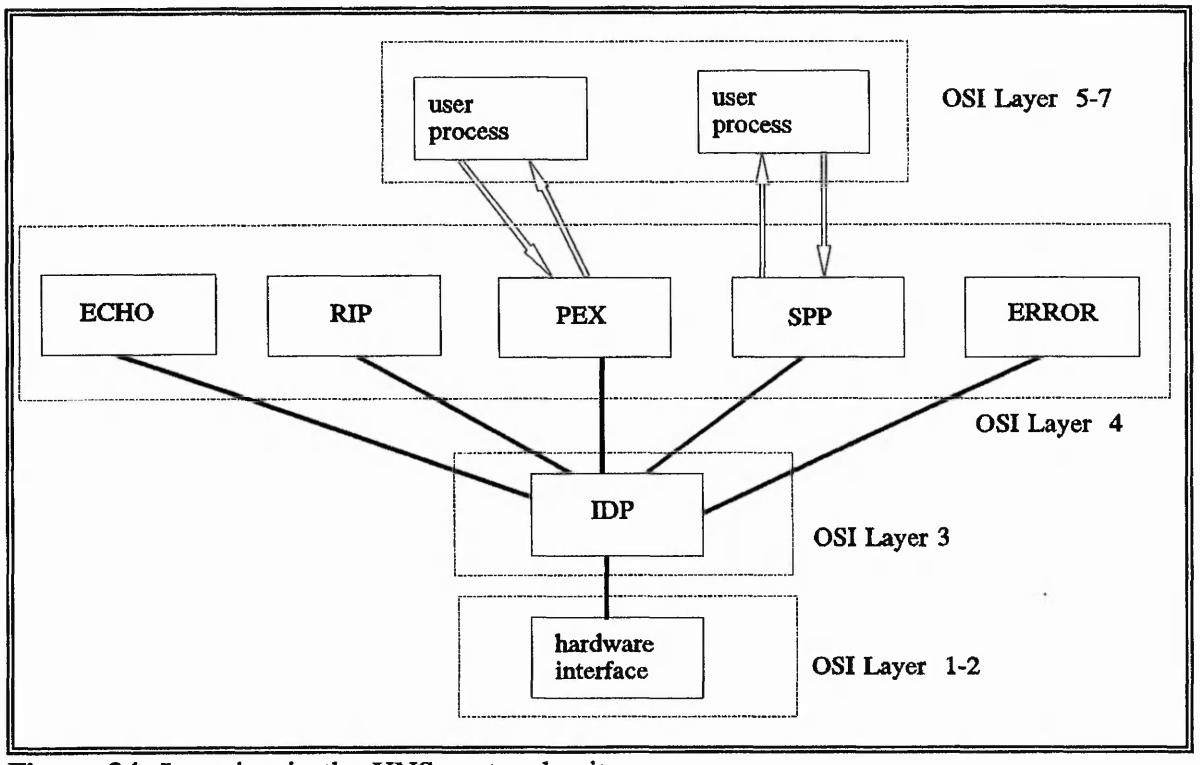

Figure 34: Layering in the XNS protocol suite.

software package . This package runs under 4.3 BSD and System V.

Amongst some of the services this package provides are: Remote Operation Service(Similar to the Remote Procedure call techniques), FTAM(Transfer of text and binary files, directory listings, file management), FTAM/FTP gateway and VT(Virtual Terminals). It is predicted that a gradual shift from non-OSI protocols to the OSI protocols is taking place, and the next major release of 4.xBSD should have support for some of the OSI protocols.

Unix-to-Unix Copy

UUCP is a collection of programs that can be used to copy files between different systems and to execute commands on other systems. Some of these programs are as follows: "uucp" program(this is different from UUCP, since UUCP represents the collection of the programs) invoked by users, copies a file from one system to another, "uux" program spools a command for execution on another system, "uucio" program

usually run as a daemon process to perform the actions that have been requested by previous uucp or uux commands, and "uuxqt" program executes files that were generated by uux.

However, the underlying fact remains that UUCP does not provide the primitives necessary for user processes to establish interprocess communication across the network.

5.3 Requirements of a distributed programming language

Ideally, programming support for distributed applications must fulfil there primary requirements. Firstly is the ability to assign different part of a program to run on different processors, this is known as "configuration" or "mapping". Secondly, the processors in a distributed system must be able to communicate with one another in an asynchronous or synchronous manner. And finally is the ability to detect and recover from partial failure of the system.

The suitability of a distributed programming language for a given distributed hardware would be measured on how many of the aforementioned requirements it can satisfy.

5.3.1 Parallelism

The underlying reason for supporting parallelism is achieving speedup through parallelism by running an application program on a distributed computing platform. Parallel applications can be classified by the grain of parallelism they use. The grain is the amount of computation time between communications. Fine-grain parallelism is most suited for tightly-coupled distributed systems. On the other hand , the communication overhead prohibits the use of fine-grain parallelism, thus rendering large-grain parallelism as the only suitable alternative for loosely coupled distributed systems.

Finally therefore, a truly parallel distributed programming language should provide the means to encapsulate the grains and allow them to communicate with one another. Moreover, the encapsulating mechanism should also be mappable to the underlying target

hardware.

5.3.2 Interprocess Communication and synchronization

Whilst a distributed programming language providing the means by which truly distributed application programs can be developed, the underlying operating system on the other hand must also provide the communication primitives(e.g. SEND, RECEIVE ...etc) for the distributed processes to communicate with each other. In the preceding sections a distributed computing system was described as "a federation of autonomous processors that do not share primary memory, but cooperate by sending messages over communication network", thus leaving the task of selecting an appropriate message passing mechanism to the programmer. Message passing can be performed either synchronously or asynchronously, as was mentioned in the preceding sections. In an asynchronous communication, the sender does not need to wait for the availability of the receiver and it proceeds with its execution after sending the message. If a reply message is expected by the sender, it can later wait explicitly for the reply. In synchronous communication, the sender is blocked until the receiver accepts the message. Synchronous communication is easier to use , but it has the potential disadvantage of reducing parallelism (i.e. no data is exchanged between the caller and the callee before a link is established, thus the caller should receive an acknowledgement before it can proceed further with its execution).

An important form of synchronous communication is Remote Procedure Call (RPC)**[15]**. An alternative method, which was first used for inter-task communication in Ada language**[1]** and later employed in other languages**[6]**, is the Entry Call(Rendezvous). The rendezvous mechanism provides a high-level conceptually consistent mechanism for interprocess communication which unlike other communication methods does not rely on explicit calls to the underlying operating system(Fig.35).

Whatever form of communication may be chosen, the operating system knows little or nothing about the content of the messages. Messages are typically regarded as sequences

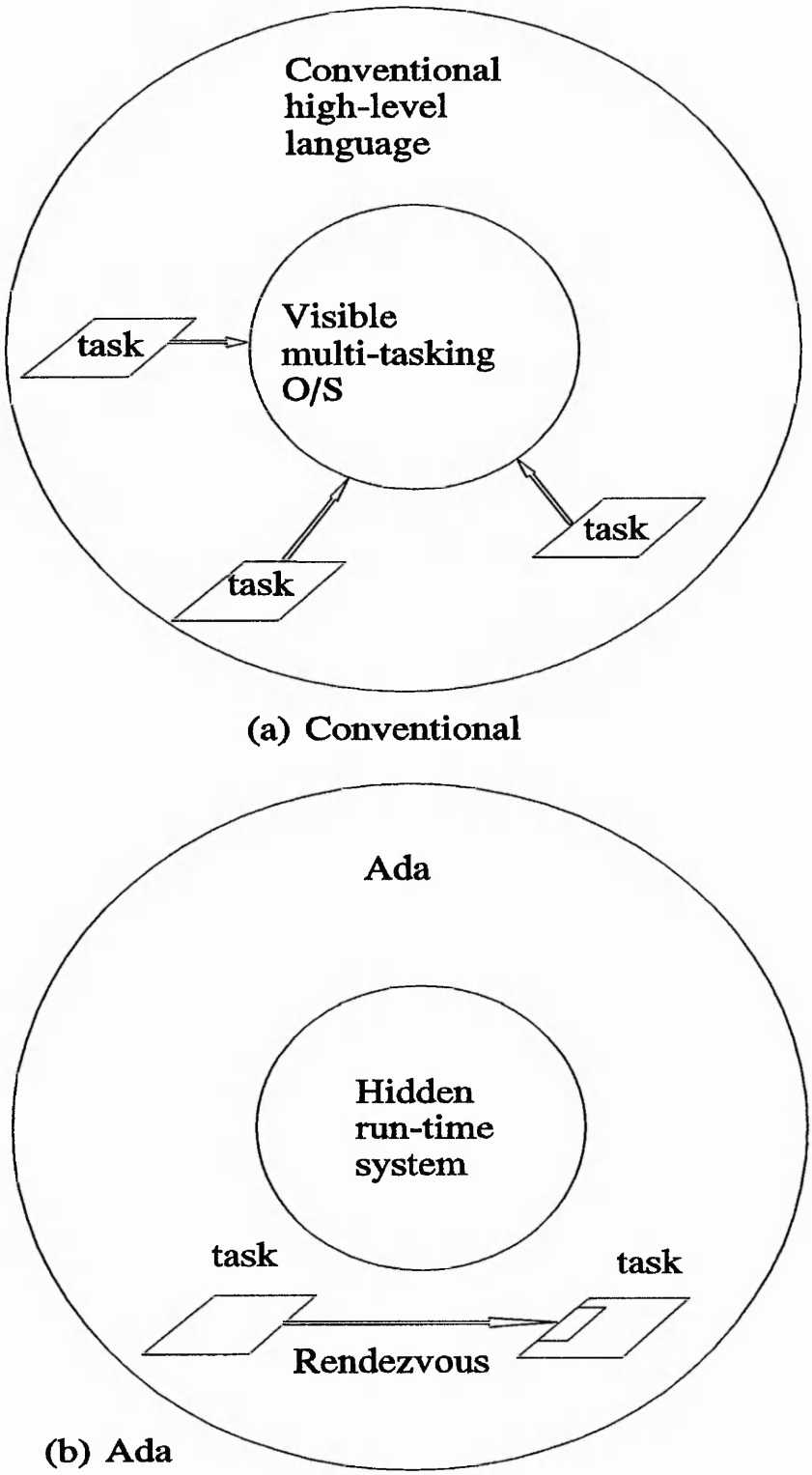

Figure 35: Conventional Vs Ada approaches to tasking.

of bytes. This imply that the sender and receiver of the message must agree on the form and contents of the message, because inconsistencies will probably go undetected.

5.3.3 Partial Failure

The programming language used to develop a real-time application, must provide some form of recovery mechanism for processor failure situations. This is known as the partial failure property. Therefore, failure of a processor in a distributed computing system should not effect the correct functioning of the other processors in the system. Amongst the programming languages that provide such a mechanism, is the Ada language with its "exception" mechanism. In an erroneous situation when an exception is raised, exception handlers written in the code earlier can handle the situation and allow the system to recover.

5.4 Programming distributed systems in ADA

Ada programming language retains certain characteristics that make it an ideal language for the development of real-time application. These include strong typing, abstraction(i.e. packages), separate compilation and fault tolerant mechanism(i.e. exceptions).

However, although the advantages of Ada for uniprocessor environment running real-time embedded applications have been well documented, its implementation of real-time application programs in distributed environment presents several problems. These problems generally stem from the fact that an application program developed as a collection of communicating processes in Ada cannot be mapped onto a distributed computing system, because the nesting and packaging features of the language allow tasks to share data. It is possible to simulate the effect of shared memory in a loosely-coupled distributed system, but this would require complex message exchanges which results in large overheads and delays in communication and response time respectively.

Furthermore, even if the sharing of data is banned amongst remote tasks, the mechanism for configuration management, structuring and design in Ada do not provide the kind of

units appropriate for execution in a distributed environment. In addition, neither packages nor tasks possess all the properties necessary for representing the required units of design and distribution. A task may not be a library unit in its own right, but must always be enclosed in some other unit (e.g. a package). Therefore, it is not convenient to treat a task as a unit of distribution, separate from the encapsulating unit containing it. On the other hand , a package does not have its own "thread of control", except that used for its initialization, and consequently does not model well a concurrent process running on a separate processor. The only construct which Ada recognizes as being independently executable is the " main program". No individual library unit can be executed outside the context of a "main program". Ada, therefore provides no single structural unit which is entirely suitable for modelling abstractions of independent network nodes. In the next sections strategies for overcoming such problems in Ada are investigated.

5.4.1 Strategies for programming distributed applications in Ada

The selection of a distributed approach is influenced by the nature of the application. Applications influence the effort spent for development and implementation of distributed systems. Expressiveness and generality of approaches will be dependent on the applications. For instance, application program developed for embedded systems, in many cases, lead to simple and effective but very restrictive solutions. Projects which provide solutions for general purpose applications have to solve many problems. For example, the implementation of remote communication mechanism.

There are various methods for developing distributed application programs in Ada**[23,4,138]**. The target architecture for distributed execution of the Ada program plays a predominant role in selecting an approach. It strongly influences the design and implementation decisions made. Every implementation relies heavily upon the underlying hardware and software design. The hardware configuration may consist of loosely-coupled or tightly-coupled systems. The approached support different storage models, like distributed memory, shared memory or global virtual memory. They use various

communication models, like message passing or shared objects.

The methodologies used for programming distributed applications in Ada, are categorised according to their architectural requirements(as mentioned above). They have emerged into four main approaches from which a strategy can be derived to suit the needs of the target architecture. These four are**[4]**: i) Distributed Target Compiler approach, ii) constrained design, Distributed_Target APSE approach**[30]**, iii) source code distribution approach, and iv) separate programs approach.

The first method would use a validated Ada compiler to generate object code for distributed targets.In this method, the Ada software is written as a single application program. Pragmas inform the compiler what processors comprise the system and where the various code units are to be allocated. The compiler then produces separate object code files for each processor in the system.The major problem with this method is that if the processors do not share a common memory, then a distributed run_time Kernel must be built to synchronize the processors.

The second approach restricts the software design so that the only communication allowed between loosely coupled processors is through the task rendezvous. The designer partitions the application program into "Virtual Nodes (VN)", the VNs are statically assigned to physical nodes (PN). The PNs may be single or multicomputer systems. The multiple processors of a physical node must be tightly coupled with a common memory to allow access to shared data. Processors mapped to a VN should be homogenous to avoid conflicts between various data representations in shared memory. Further requirement of this approach is for the Ada Programming Support Environment (APSE) to incorporate automated construction tools to automatically generate object code suitable for the target system.

The third method partitions the Ada source code and compiles for the specific targets with existing compilers. The objective of this method is to permit multicomputer programs to be developed in the Ada language with a minimum of design restrictions.

The multicomputer application is developed as a multitasking Ada program on the APSE. Thus, the application is developed and tested independently of how the software is distributed on the target hardware. The tested multitasking software is then partitioned for the multicomputer system architecture. This process may be automated with a tool called the Automated Software Partitioner(ASP). The distributed software may be tested on the APSE by placing the main program for each processor in an associated task and importing a package written to simulate the communications network of the target architecture.

The fourth technique uses somewhat more traditional method. This involves the design and development of separate source programs for each processor in the distributed system. The interprocessor interface software is explicitly coded in each application. In the first three approaches a single Ada program is partitioned onto its functional boundaries and each section is mapped onto a processor in a distributed system. On the other hand, however, in the fourth approach an Ada program is provided for each processor and the collection of programs constitute the application software.The comparative study of the four methods explained above , indicates that no one approach can be identified as the universally better approach than the others. Rather, conditions surrounding a particular application will suggest the best approach for that application. The fourth method for example ,"A program per processor",has the drawback of not conforming with the intended use of Ada. This is the result of extending the program boundary beyond that of a single Ada program, therefore losing software portability. Furthermore, the data transfer between processors is confined to procedure and function calls when other Ada language features(such as Rendezvous) is a more appropriate means of communication.

The basic characteristic of the first three approaches, on the other hand, is that the application software is viewed as a single Ada program, distributed across the target system. The main advantage of this method is that all interfaces between the distributed program fragments can be type checked by the compiler.

Within these approaches two general strategies can be identified: post-partitioning and pre-partitioning.These partitioning schemes will be explained in the following sections.

5.4.1.1 Pre-partitioning scheme

With the view to using standard Ada compilers, the approach of choosing a specific Ada construct as the only possible partitioning unit is the most attractive one. However, no one unit provides adequate resources for partitioning an Ada program for execution on a distributed system, due to the reasons mentioned in the preceding sections. New strategies had to be devised to deal with the shortcomings associated with distributing Ada programs. Furthermore, these approaches must conform to the context of the current language definition.

The "virtual node" approach for partitioning a program for distributed execution has been the subject of discussion in both academic and industrial circles**[147]**. The working group responsible for development of such strategies acknowledge that the virtual node approach is quite useful since it requires limited changes to the language(i.e. imposes limited restriction on use of some of the features of Ada). They accept, on the other hand, the criticisms regarding the partitioning restrictions imposed on the users of the methodology**[147]**.

A virtual node is a collection of library units and a unit of distribution for Ada programs. Virtual nodes could be allocated to a single physical node but a single virtual node could not be allocated to more than one physical node. The granularity of a distributed program is defined by its virtual node components. The virtual node approach rules out task as a distributable unit since it can not be a library unit, however, a task can be a component of a virtual node.

The communication between virtual nodes is restricted to either remote rendezvous or remote procedure call(RPC) or both, thus ruling out data sharing. It is possible, however, for two virtual nodes to have units in common. In these cases, the library unit must either be the interface unit of other virtual nodes or be templates, in which case they can be replicated on each virtual node without violating their semantics. A template library unit

is one which does not have a global state(e.g. generic packages with only type declarations or procedures or tasks which do not access global memory). Finally, the application program is viewed as a network of communicating virtual nodes.

5.4.1.2 Post-partitioning scheme

In post-partitioning scheme, the application design process is divided into two phases: functional design (application program) and functional distribution (hardware mapping) **[24,73,77,74]**. The partitioning information and source code are usually kept separate. Strong requirements are put on the run-time system for efficiency,especially when high fault tolerance is required.

This strategy is based on the hypothesis that all Ada entities (data or control) can be distributed. Therefore, this enables an initial program to be designed without any restrictions. This approach is attractive, but it is also the most complex to implement. This is due to the fact that, mechanisms guaranteeing the coherency between different copies of variables shared by the various modules, distributed throughout the distributed system, have to be foreseen. Hence a specific distributed operating system capable of providing such mechanisms must be available.

The method of partitioning on any part of an Ada program is developed in the APPL project **[25]**. The aim was to supply an application independent system that supports the execution in a distributed environment. The functional mapping of the application is described in a separate language, APPL(Ada Program Partitioning Language). The development of a system may start on a uniprocessor system and may later, in the final integration phase, be transferred to the distributed target. The APPL approach gives the application no knowledge of the distribution. Hence, all fault tolerance has to be implemented within the underlying run-time system.

5.4.2 Object access in distributed systems

The structure of Ada permits two different modes of access among execution objects(subprograms or tasks). One is by passing parameters in subprogram calls or task entries. The other mode is by shared variables that exist in the common scope of the execution objects.

Ada requires that parameters be passed by copy. To avoid possible inefficiencies parameters that are arrays, records or task types may be passed by reference provided the effect is by copy. In the case of execution objects on tightly-coupled machines passing by reference, while keeping the appearance of by copy, can be efficient. However, in the case of loosely-coupled machines, passing by reference will lead to memory address error. It is natural therefore to pass all parameters by copy. On the other hand, communication between two execution objects through shared variables can be most naturally implemented with a shared logical memory.

Tightly-coupled machines

In the case of tightly-coupled machines, shared data object references can be implemented as in a uniprocessor case. This requires that the underlying hardware memory protection system allow user processes on multiple machines to access the same regions of physical memory, but otherwise creates no problems for handling variables or pointers not already present in the language. Access to remote execution objects requires a signalling mechanism among the processors involved to permit the receipt of a remote call, but requires no special mechanism for handling the actual parameters of the call. The trade-offs between communication by shared variables or message passing are the same as for a uniprocessor implementation.

Loosely-coupled machines

In a loosely-coupled architecture, significant differences exist between shared variables

and message passing communication. Each shared variable reference to a remote data object must be translated into a remote procedure call to a server process on the processor holding the object. This server must perform the required operation, and if necessary return a message containing the value of the object. On the other hand, if the variables are communicated via message passing, references to them will be to the local copies and communication overheads will be substantially less.

5.4.3 Virtual Node approach

The post-partitioning approach for distributing Ada programs(e.g. APPL[25]), makes a clear distinction between the programming phase and the program partitioning phase. This method therefore allows an application program to be developed as a single program using the complete range of features that Ada has to offer. This means that programs can be developed using traditional methods for designing software for uniprocessor systems. A specific methodology for distributed system is not required.

However, a consequence of using such methods is that the run time system must support remote variable updates. This is most complex to implement since mechanisms guaranteeing the coherency between different copies of variables shared by the various modules distributed throughout the distributed system have to be foreseen. Hence a specific distributed operating system capable of providing such mechanisms must be available. Furthermore, while the application can be debugged using off the shelves standard debugger for uniprocessor systems, testing the program on the distributed hardware would require a distributed debugger. In addition, distributed application programs may look different from application programs that are not distributed and the software designer will have to know a priori if the target is distributed or not. If this is the case then the benefits of this approach are greatly diminished. Finally, this approach requires a considerable investment in tools to support software development for distributed targets.

In view of the shortcomings associated with post-partitioning, it was decided that pre-

partitioning would offer a more cost effective alternative compared with post-partitioning. In the pre-partitioning scheme a particular Ada construct is selected as the sole unit for partitioning throughout the design and programming process. The programmer is obliged to accept any constraint the choice of construct entails. The notion underlying this strategy is that of "Virtual Node", which is an abstraction of a physical node in the distributed system**[71,132]**.

The essential properties of a virtual node are as follows:

- Its internal state is invisible to the other nodes.
- It has activities(control flows), which consists of local actions communication and coordination actions with the other nodes of the system.
- It communicates with other nodes through well defined interfaces and according to message like transmission protocols(communication by shared variables is excluded).
- It can be executed autonomously on physical nodes of the target system or be incorporated in a group of virtual nodes to form a new executable program.

Furthermore, it must be possible to define a virtual nodes configuration by explicitly indicating the connections between these nodes, thus enabling the separation of component programming phase from that of the definition of the final system's configuration. The interfaces of the virtual nodes are then described through their communication ports with the outside world, and a system configuration can then be defined using the external interfaces and the specification of the ports connections.

The notion of virtual node is found in most languages which have been designed with the intention of supporting distributed programming(e.g. the "guardian" of Argus**[84]**, the "task module" in conic**[127]**, and the "processor module" in starmod**[22]**).

5.4.3.1 What should Virtual Nodes represent.

A virtual node can represent a number of constructs in Ada, these objects range from an entire program in one extreme and extend to single tasks or simple packages in the other

extreme. Several projects adopted the virtual node approach for developing distributed applications. The Michigan Distributed Ada**[139]** project uses library packages and subprograms as the unit of distribution. Application programs are specified as sets of cooperating virtual nodes, where each virtual node is a collection of library packages and subprograms. The Michigan Distributed Ada (MDA) project allow the specification of distributed program by the special pragmas to indicate where each object should reside in the distributed network. The MDA project, like the Honeywell Distributed project**[25]**, adopts the post-partitioning approach to develop distributed programs.

The SD-Ada Multiprocessor System**[21]** distributes the Ada tasks of a program amongst the processors of a tightly-coupled system. However, making tasks the unit of allocation would have a number of problems. The first problem is basically that of task identification when they are allocated to different processors, while belonging to the same program (unique Ada pathnames should be specified for every task). This can result in tasks on one processor being statically dependent on tasks on another processor. In addition, any data visible to that program must be located in a shared memory visible to all processors involved. A solution is defined in the form of a "library unit" to be the unit of allocation to a processor. Any task objects that are declared within the library package specifications are created when the task declaration in the corresponding package specification or body is elaborated. Hence by controlling the processor that elaborates these "active" library units, the user is able to control the allocation of library tasks to processors. The library tasks are therefore the only tasks over which the user has any control; all other tasks execute on the same processor as the task that creates them.

The post-partitioning approach of MDA and Honeywell distributed Ada, will not be considered further because of the shortcomings that are generally associated with the post-partitioning schemes(as mentioned before). Moreover, the SD-Ada project is targeted at tightly-coupled distributed systems thus making it unsuitable for loosely-coupled systems studied here in this report. The remaining part of this section is devoted to methodologies which rely on the notion of virtual nodes for programming distributed loosely-coupled systems in Ada. The partitioning scheme employed in these methods is that of the pre-

partitioning approach.

The first of these methods studied here is that of ASPECT-York Distributed Ada[**68**] project. From amongst language constructs offered by the Ada language, packages are chosen to represent virtual nodes. They satisfy most of the conditions set for a language construct to be effective as a virtual node[**71**]:

- separate compilation and library units, and
- exception handling facilities to cope with communication failures.

However, packages are static units, this means that dynamic instantiation of packages is not possible. Limitation had to be imposed upon the sharing of library packages since if one instantiation of the library unit exists in the system, all procedure calls,entry calls, and variable updates associated with that unit can potentially occur from a remote site, thus violating the interface conventions between virtual nodes(mentioned in the preceding sections). Another subject examined by the project is the inter-virtual node communication. The underlying question was whether to use a remote version of Ada's inter-task communication mechanism(rendezvous) or the remote procedure call (RPC) mechanism adopted by other projects[**15,82,93**]. The RPC was chosen since the ASPECT project provides multi-language support and the only well known mechanism for transferring control between virtual nodes(written in other languages) is RPC as compared with remote rendezvous.

The second and final method is that of the DIADEM project[**5**]. The DIADEM project examined two constructs that Ada provides which may be used to represent virtual nodes. These were "tasks" and "packages". Tasks could not represent the unit of distribution since Ada's scoping rules allow them to share data thus violating the virtual node rules on "no data sharing allowed unless defined within a template area"(as mentioned before). Furthermore, tasks can be library units since they must always be enclosed in some other unit(e.g. packages). These shortcomings therefore make tasks unsuitable for distribution. A package, on the other hand, does not suffer from these shortcomings, but lacks its own

"thread of control" and consequently does not model well a concurrent process running on a separate processor. Furthermore, reconfiguration would not be possible, without shutting the whole system down. This is due to the fact that packages are static constructs and can not be dynamically instantiated.

Acknowledging these shortcomings associated with tasks and packages, it was concluded that Ada does not provide any single structural unit which is entirely suitable for modelling abstractions of independent network nodes.

The solution adopted in DIADEM is to define the required distributed components as independent Ada subsystems called virtual nodes. The subsystems represent the highest level units of problem decomposition and although they can be arbitrarily complex structures, from the point of view of distribution they are logically atomic units that can only sensibly execute on a single machine. They are thus strongly cohesive entities grouping together intimately related objects, but having minimal interaction with other virtual nodes. However, the preferred method of inter-virtual node communication in DIADEM project is the "remote entry call".

5.4.3.2 Remote Communication

With the exception of shared variables, there are two methods available in Ada for expressing communication between virtual nodes, they are "procedure" and "entry" calls. The RPC mechanism, adopted by the ASPECT-YDA, can be supported by introducing additional Ada code in the form of stubs which interface to the underlying communication system. A call from a client in one virtual node to a server in a virtual node located on a different physical processor is carried out by means of the intermediate mechanism shown in Fig.36. The client and server stubs are produced as transformation of the virtual node root package specification. The client stub is a package body which replaces the virtual node root package body of a remote virtual node, while the server stub is a complete package which is placed on the processor holding the virtual node.

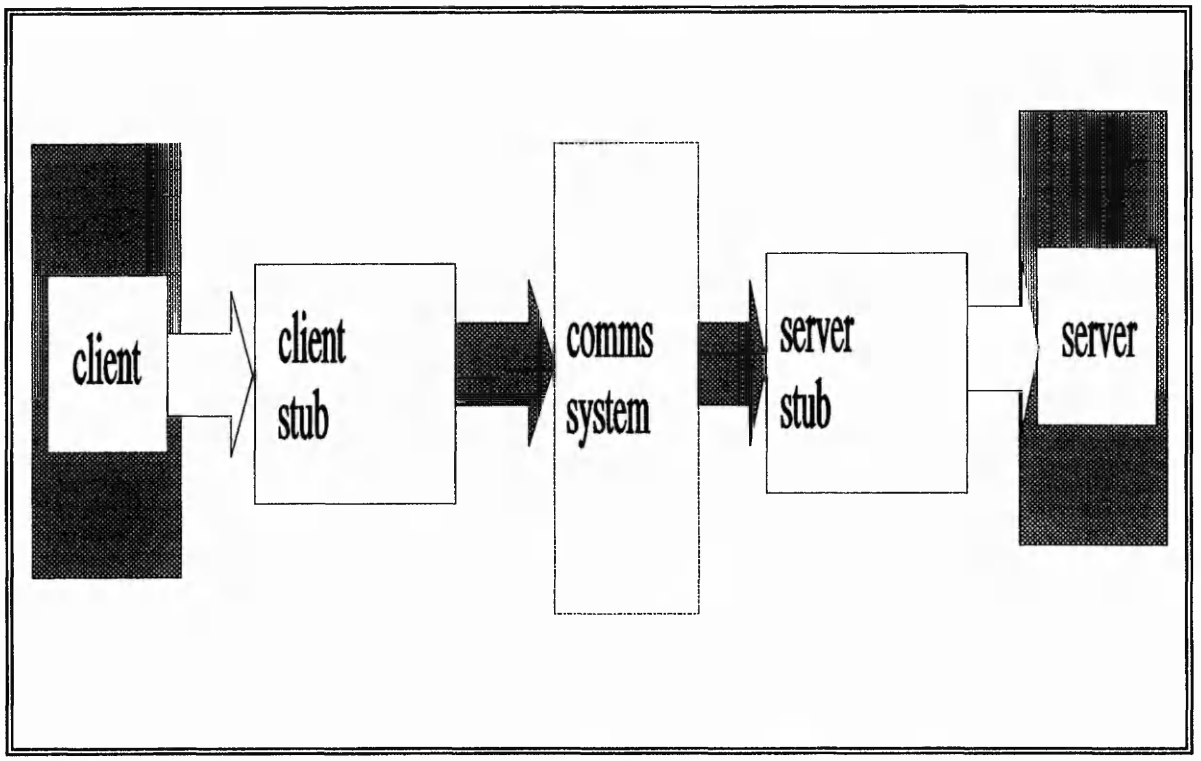

Figure 36: The Remote Procedure call mechanism.

5.4.3.2.1 Remote Procedure Call

A remote call starts as a local subprogram call from the client to the client stub, which packs the call parameters and call identification details into a record then passes this to the RPC mechanism. This constructs the call message, sends this message to the processor on which the virtual node resides, and suspends the client task awaiting the arrival of the result message corresponding to that call.

At the called processor, the server stub forms a template to create a server task which acts as the thread of control to execute the incoming call. This stub task unpacks call parameters and passes them to the RPC mechanism to return to the caller. If an unhandled exception occurs during the call execution then this is caught by the server stub and passed back to the client stub for propagation. Once the results or unhandled

exceptions have been sent back to the caller the server stub task terminates. As this server task is unknown to the user's Ada program there are some optimisation which reduce the overhead normally associated with Ada tasks; for example no server task ever has any entries. A user created task must leave some information when it has terminated as reference may be made to it from elsewhere in the program. Server stub tasks are not known by the original program and so have no such requirement, so there is no permanent memory overhead associated with servicing a RPC request.

When the result message arrives back at the calling processor, it is passed back to the client stub, which is reactivated. This unpacks any returned values and returns them to the client, unless there was a unhandled exception during the call, in which case it re-raises the exception in the client.

The use of dynamically created server stub tasks as thread of control causes parallel execution of RPC requests, so maintaining the Ada semantics for simultaneous subprogram calls from different tasks.

The syntax of entries are very similar to procedures. The differences between them lie in the underlying protocol by which concurrent calls are selected and processed, the context in which the called code is executed, and the fact that with entry calls, the caller is able to "optout" after a certain time period has elapsed. This last difference is the most important when it comes to the implementation of remote calls.

From the point of view of the message exchanges needed to conduct a transaction, procedure calls behave in a similar way to simple entry calls because there are no extra timing signals needed to handle the possibility of the caller timing-out. In other words, remote procedure calls effectively represent a subset of Ada's three possible types of remote entry call. Therefore, an implementation of remote entry calls is likely to be able to support remote procedure calls with very little, if any, modification. The reverse is certainly not true.

The communication system developed in DIADEM is able to support both remote entry call and re-entrant remote procedure calls. Therefore, the choice between them is based mainly on user-oriented issues. The remainder of this section is devoted to the explanation of the remote entry call and how it was implemented in DIADEM, since the selected method of inter-virtual node communication in this report is remote entry call.

Having chosen rendezvous as the sole means of communication between virtual nodes, in order to achieve communication transparency(for flexible system configuration) it is necessary to extend the rendezvous mechanism to support remote transactions. Fig.37 shows the communication layer and how it relates to internationally accepted standards for network communication such as ISO reference model. In the ISO model the lowest layer providing end-to-end services is the transport layer. Therefore, implementation of the remote rendezvous end-to-end protocol is naturally handled by a layer of software built on top of the transport layer. However, due to variations of services offered by different transport layers , a software layer known as "standardising layer" is introduced to smooth out these variations and hide the special characteristic of a particular communication system.

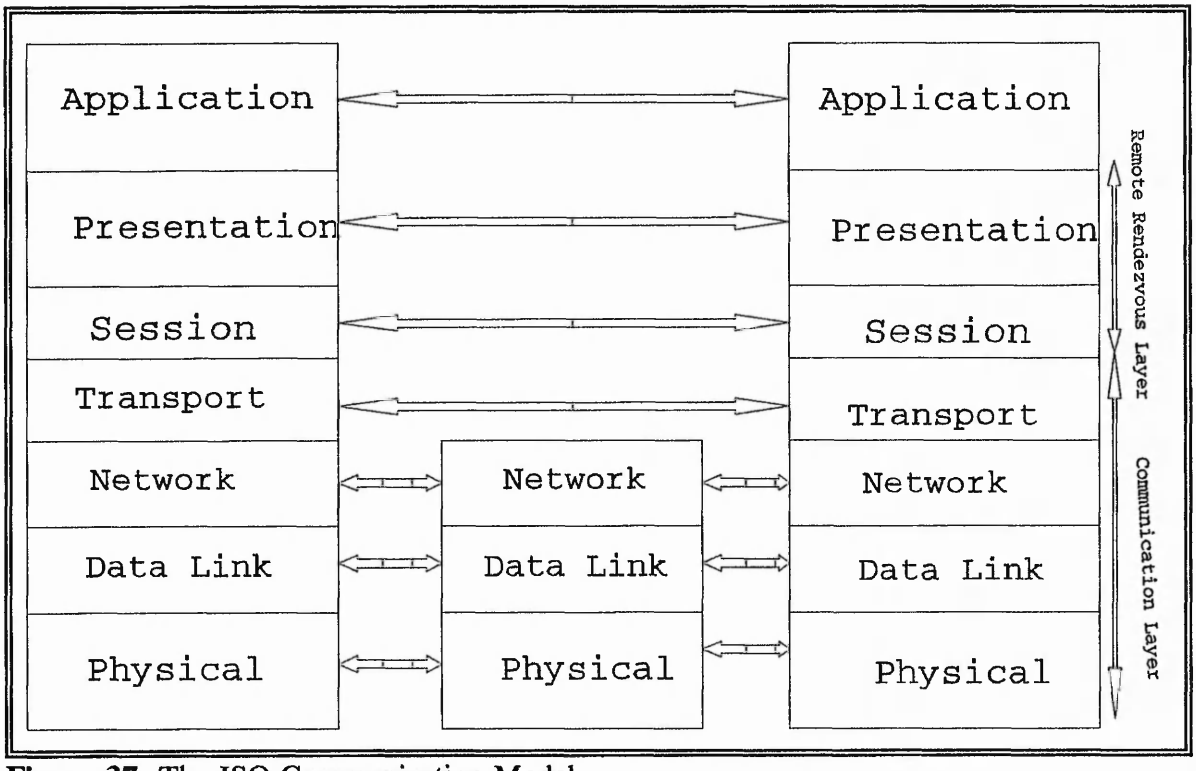

Figure 37: The ISO Communication Model.

The standardising layer, providing the interface between the host communication software and the Ada application software, may well contain processes external to the Ada "world". In such circumstances, it is the responsibility of the Ada compiler and linker to combine Ada and non-Ada software within an Ada program. This is achieved by the use of an INTERFACE pragma which notifies the Ada compiler that the body of a library unit is written in another language.

5.4.3.2.2 Remote Entry Call

The primary objective of the remote rendezvous mechanism is to reproduce, on the callee's node, the conditions which would have existed had the caller not been remote. This is achieved by creating a surrogate task to make the entry call on the caller's behalf(Fig.38).

Each virtual node which can serve as a callee in a transaction has a permanent task which listens at a prearranged communication port for incoming messages signalling an entry call. When this so called entry port task has determined which entry the call is intended for, it generates a new local agent task(surrogate task) to issue the call on behalf of the remote caller. The called task therefore, experiences precisely the same call it would have experienced had the caller not been remote. On completion or failure of the call, the local agent task communicates the result directly to the caller and then terminates.

One remaining issue was the buffering of data,"should the **in** and **in out** parameters for the rendezvous be sent to the callee's node?". This meant that all the **in** parameters would be incorporated in the first message sent to the callee to be stored until the call is accepted. This meant storage space had to be provided on the callee's end. Alternatively, data could be stored on the caller's end until the entry call is accepted by the callee. The latter approach is clearly more efficient, however, this extra efficiency in parameter buffering is gained at the expense of more message exchanges. This extra complexity introduces a delay between the time when the call is accepted, and the start of the

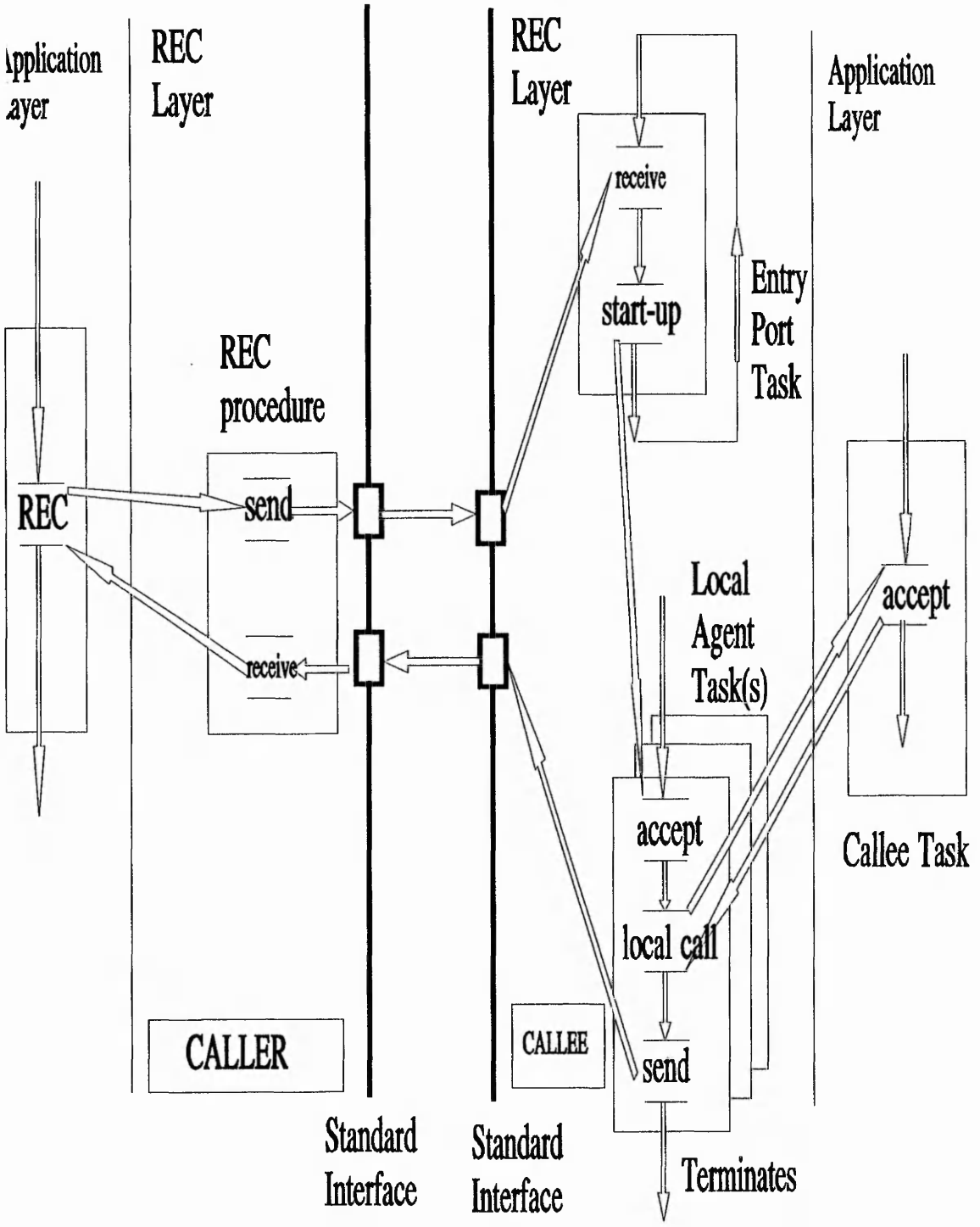

Figure 38: Structure of the Ada Remote Rendezvous Layer.

rendezvous - an inefficiency that is present for all entry calls.

The DIADEM project adopted the former method(Fig.39), thus avoid risking the extra complexity and time delay involved in the latter approach(Fig.40).

Requirements of the standard interface

The first requirement, essential in all remote communications, is to support the dynamic creation of "communication end points" or "ports". Two types of ports are required, one for sending messages, and the other for receiving them. Where ports have to be explicitly identified in the remote communication layer, they must be associated with remote system independent names. For a single remote rendezvous transaction, four uni-directional ports are required, one of each type(incoming and outgoing) for both the caller and callee. Moreover, since Ada tasks may be generated dynamically, it is also necessary to be able to create and close communication ports dynamically.

In conjunction with ports, two primitives are required for transferring messages between ports; the synchronous(i.e. non-blocking) SEND primitive which is used to send a message to a single destination and the synchronous(i.e. blocking) RECEIVE primitive which may receive messages from any source.

There are various computing environment within which the standard interface must be implemented, however, the distributed system of interest in this report consisted of a network of workstations running a general purpose multitasking operating system(e.g. UNIX,VMS). Take the UNIX operating system, for example, it supports a wide variety of services ranging from datagrams to virtual circuits(as mentioned before). Moreover, a local area network based on communicating UNIX machines provides complete communication transparency, with all the message routing and naming functions handled automatically. Thus, the configuration of the network can be completely hidden from the application software with little difficulty.

Standard Interface Layer

Fig.41 shows the package STANDARD_COMM which establishes the DIADEM standard interface to the physical communication layer. This package forms part of the pre-defined environment of every virtual node library. Its body written by the system programmer, would determine how the communication layer is constructed for the target network. As can be seen from the diagram, the package STANDARD_COMMS defines three main types. The first, SN_ID, is the type of the software node identifier assigned to each of the software nodes in the system. The other two are private types since they are not intended for direct use by the remote rendezvous layer. The first of these, HEADER_TYPE is used for the field at the head of each message packet. This contains essential system-dependent routing information. The second, RETURN_ADDRESS, is used to define a field in call packets received by entry port task. This field contains the network address of the port to which the answer message should be sent.

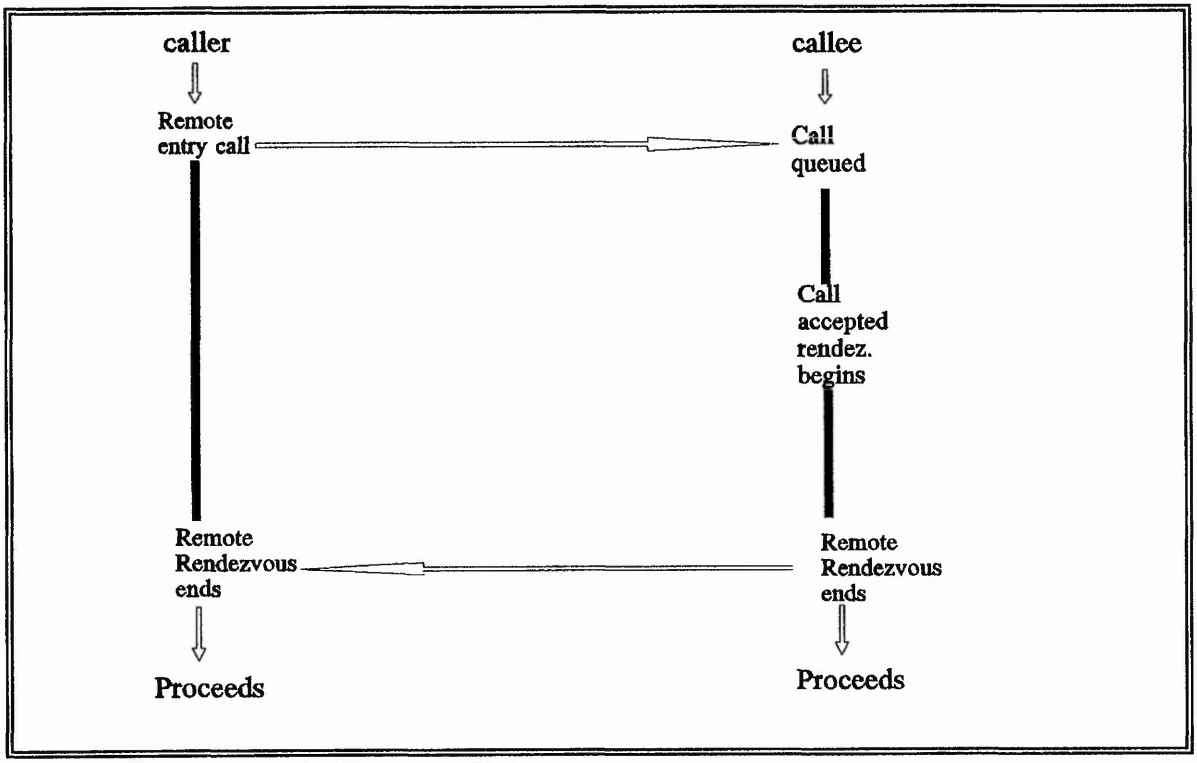

Figure 39: Parameter Buffering at Callee's end.

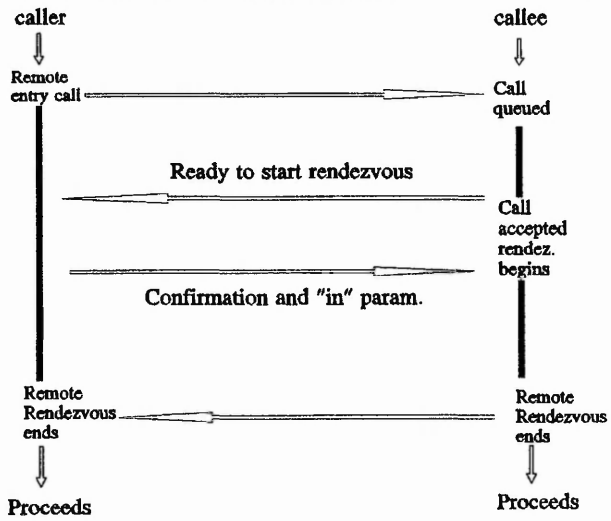

Figure 40: Parameter Buffering at Caller's end.

```
with "pakage defining system_dependent type ADDRESS_STRUCT - HEADER"
use ...
package STANDARD_COMMS is
   type SN_ID is new NATURAL;
   type RETURN_ADDRESS is private;
   type HEADER_TYPE is private;
   generic
    ...
   package CALLER_PORTS is
      generic
       ...
       package PRIMITIVES is
          ..
          procedures : ALLOCATE(..)
                       DEALLOCATE(..)
                       SEND(...)
                       RECEIVE(...)
          end PRIMITIVES;
          ..
     end CALLER_PORTS;

     generic
       ...
     package CALLEE_PORTS is
       ...
       procedures: ALLOCATE(..)
                   DEALLOCATE(..)
                   SEND(..)
                   RECEIVE(..)
       ..
     end CALLEE_PORTS;
   private
     ...
end STANDARD_COMMS;
```

Figure 41: The standard interface package.

STANDARD_COMMS contains two generic packages - CALLER_PORTS and the CALLEE_PORTS. Instantiation of one of these packages causes the generation of two logical ports according to the model illustrated in Fig.42. CALLEE_PORTS is instantiated in the entry port task of each software node, and generates one static receiving port and one sending port. These both exist for the life-time of the software node, and the receiving port must reside at a fixed network address. The correspondence between this network address, the software node identifier and the physical node is stored in the network name table(s). The package CALLER_PORTS, on the other hand, is instantiated in the declarative part of each internode calling unit and causes the generation of a similar pair of ports. In addition, these packages define a range of primitives associated with the logical ports.

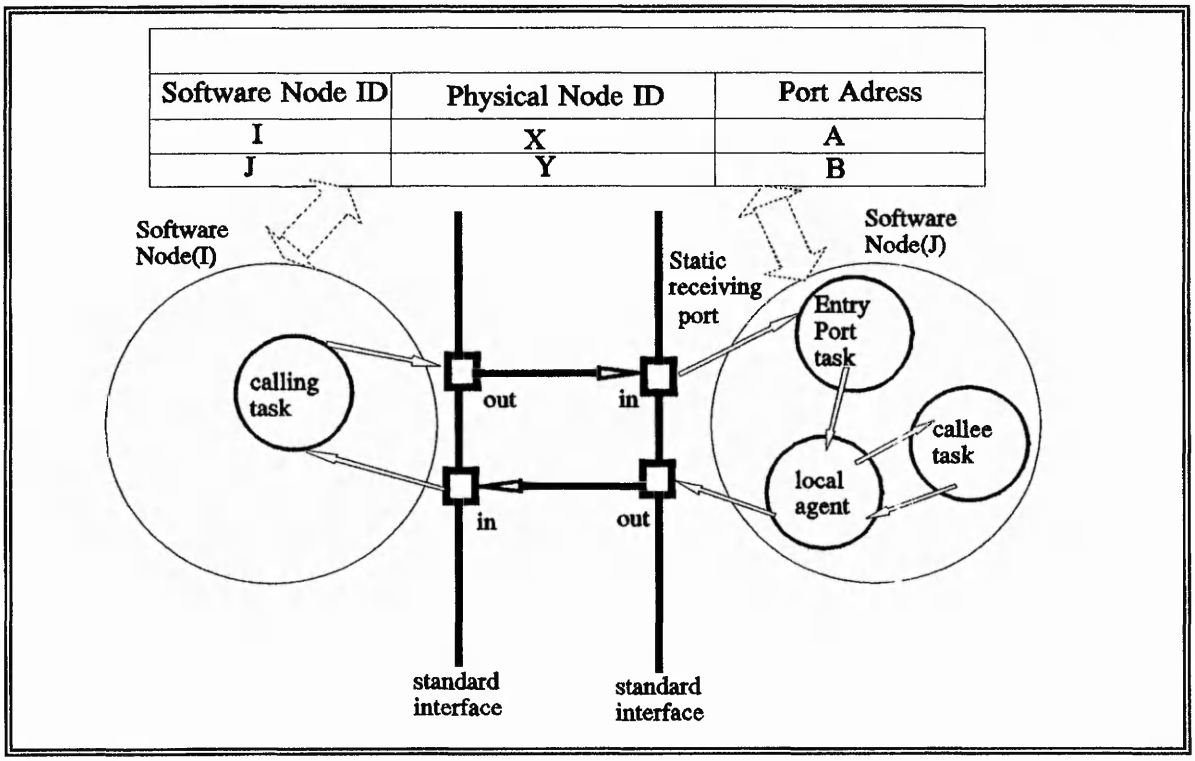

Figure 42: The Communication Model.

Implementation Issues

The package STANDARD_COMMS is designed specifically to be implemented on top of most commonly used type of communication systems. The system of interest here, is based on a network of SUN workstations connected by Ethernet, and hence involves implementation of the standard interface on top of the Berkely UNIX 4.3 BSD system provided with the SUN workstations.

The inter-process communication facilities provided by UNIX are based on **sockets.** There are many different kinds of sockets and associated primitives supporting a wide range of services. Two standardized communication domains, are currently available - the UNIX domain(AF_UNIX) and the Internet domain(AF_INET). The former of these is only suitable for communication on single machine, but using the latter it is possible to make the distribution of processes over the network completely transparent to processes themselves.

The Internet domain protocols are comprised of connection-oriented and connectionless protocols.

Connection-oriented protocol

Fig.43 shows a time line transaction that takes place for a connection-oriented protocol. The server process starting first, creates a socket end-point whose address is defined by a record type template. The "bind" system call subsequently uses the address structure of the socket to construct a unique system wide name for the socket(i.e. the server process calling the bind system call registers its address with the underlying communication system, indicating that: this is my address, any messages received for this address must be passed to me). The "listen" system call then indicates that the socket is willing to receive connection. The listen call is immediately followed by an "accept" call which takes the first connection request on the queue and creates another socket with the same properties as previous socket. If there are no connection requests pending this call blocks the caller until one arrives. If a call is accepted however, a read is performed on the

ports. An acknowledgement is sent to the client if the read system is successful.

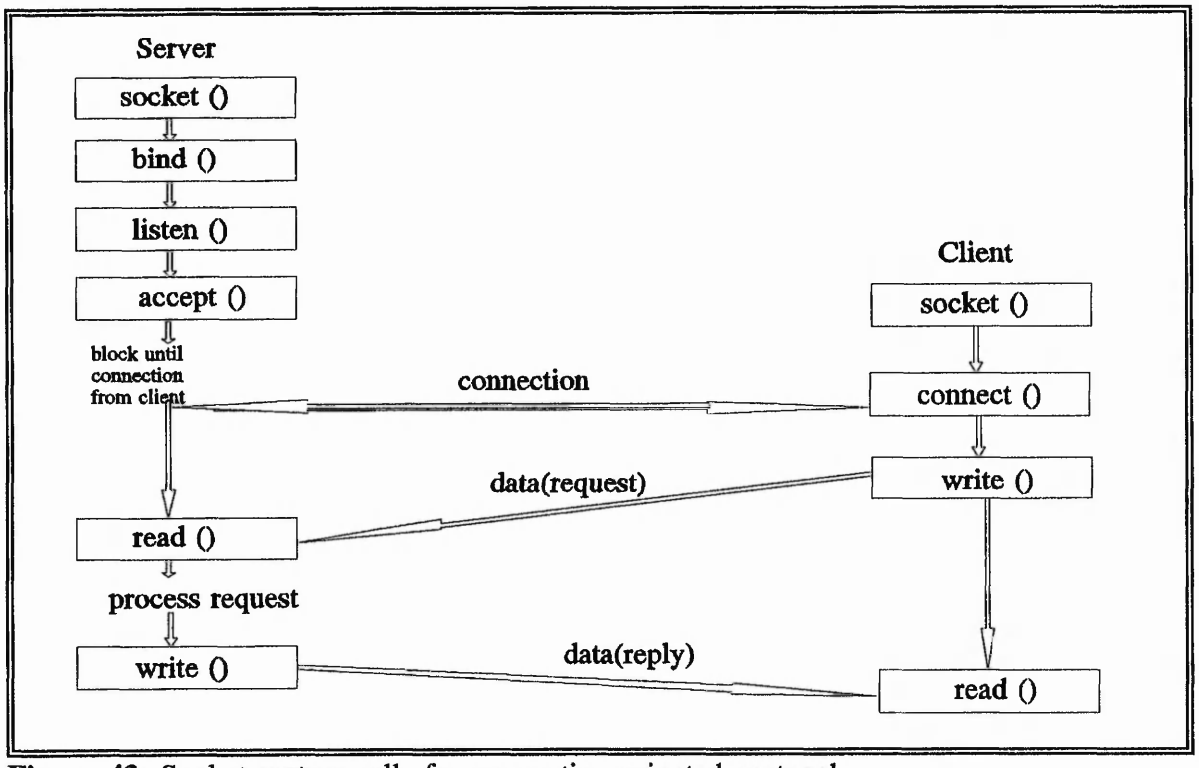

Figure 43: Socket system calls for connection-oriented protocol.

Connectionless protocol

For a client-server using a connectionless protocol, the system calls are different from that of connection-oriented protocols. Fig.44 shows these system calls. The client does not establish a connection with the server. Instead, the client just sends a datagram to the server using the "sendto" system call, which requires the address of the destination(the server) as a parameter. Similarly the server does not have to accept a connection from a client. Instead, the server just issues a "recvfrom" system call that waits until data arrives from a client. The recvfrom returns the network address of the client process, along with the datagram, therefore the server can send its response to the correct process.

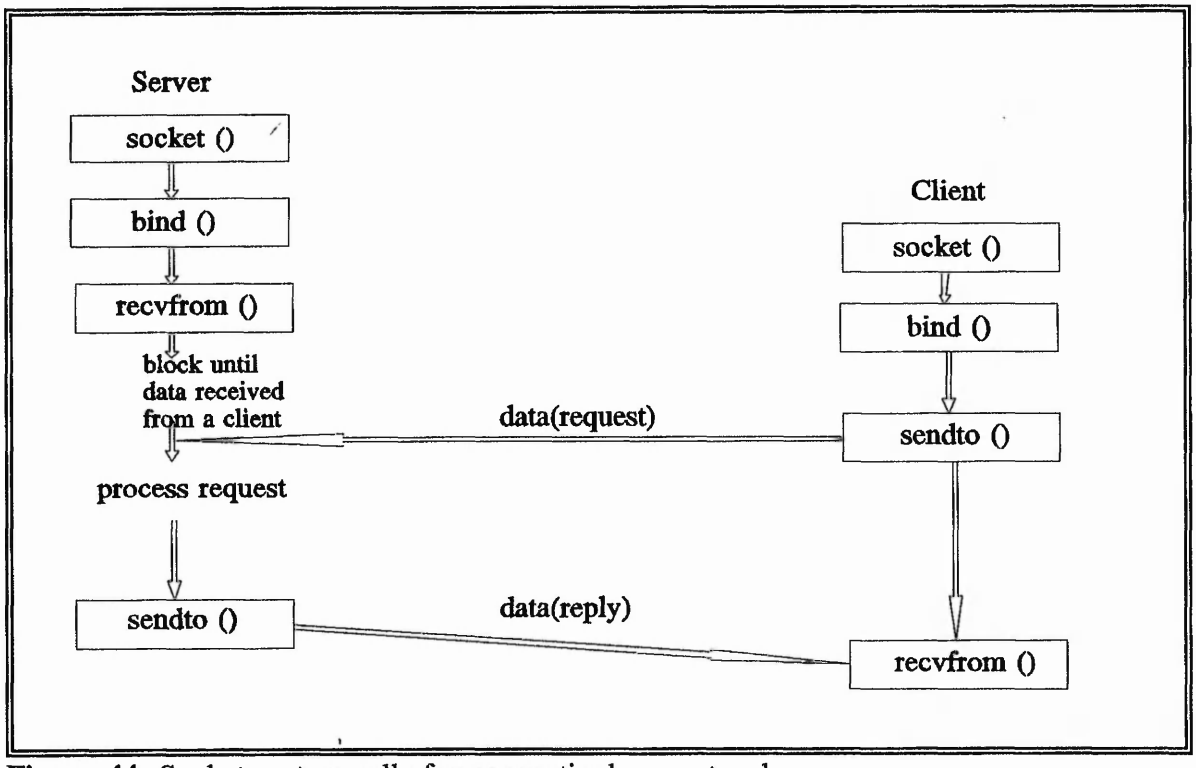

Figure 44: Socket system calls for connectionless protocol.

Socket addresses

The BSD networking system calls require a pointer to a socket address structure as an argument. For the Internet family the address record type(structure in C) is shown in Fig.45. Moreover, since the socket address is used by the BSD communication layer, it must be formatted into fields of bits, each field defining different component of the socket address record. Such record blocks can be modelled in Ada by a record representation clause which associates each record component with specific location of its bit field in the block of data. Since each component of the record occupies certain number of bits , it is important to have some means of determining the position of the record components.Furthermore, the alignment of the component of the record type may help to access the data quicker. In Ada the position of record components is aligned according to the number of "storage-units" they occupy. The storage-unit size differs from one computer to the next, in our implementation however, storage-unit size is given in the predefined package SYSTEM as 8bits. On the other hand, if this record is to be used by

```
type IN_ADDR is
 record
  s_addr    :INTEGER_32;
 end record;

type SOCKADDR_IN is
 record
   sin_family    :INTEGER_16;
   sin_port      :INTEGER_16;
   sin_addr      :IN_ADDR;
   dummy1        :INTEGER_32;
   dummy         :INTEGER_32;
 end record;
```

```
for SOCKADDR_IN use
 record at mod 8;
  sin_family    at 0*word range 0..15;
  sin_port      at 0*word range 16..31;
  sin_addr      at 1*word range 0..31;
  dummy1        at 2*word range 0..31;
  dummy2        at 3*word range 0..31;
 end record;
```

Figure 45: Socket address Structure and Low-level mapping.

Bits and Bytes for Sun/SPARC systems.

Byte n	Byte n+1	Byte n+2	Byte n+3
31 32 ... 24	23 22 ... 16	15 14 ... 08	07 06 ... 00

Figure 46: Bytes and Word length of the target system.

the communication layer, the byte order of the record type must correspond to the byte order of the underlying system. In this case(i.e. network of SUN workstations) the bit and byte ordering is as shown in Fig.46. Using the given byte ordering format the final address record representation clause is as shown in Fig.45.

5.4.3.4 Virtual Node structure

The preceding sections introduced the concept of virtual nodes as an important structuring method for supporting the development of flexible distributed program. A number of Ada constructs were examined which may represent the virtual node in a distributed system, unfortunately none of them is entirely suitable for representing virtual nodes(as mentioned before). They each possess some of the required properties, but not the necessary combination.

Moreover, the problem of "thread of control" in packages may be solved by an encapsulated task(i.e. an active package). However, the problem of dynamic allocation of packages can only be overcome if Ada langauge is extended(e.g. package types).

In view of difficulties of representing a virtual node any single Ada construct, the DIADEM project has chosen a more general solution**[5]**. In DIADEM virtual nodes are modelled by a collection of library units connected in the normal way to form complete program structures. This has two important advantages: it allows virtual nodes to be designed as complete subsystems, with all the complexity possible in a normal Ada program, and it means that virtual nodes are no longer limited to presenting only a single interface to the rest of the system, but may define as many interfaces as appropriate for the desired communication pattern.

Furthermore, no additional instructions(e.g. pragmas or formal comments) are added to the units in a virtual node library in order to allow visibility or sharing of other library units. Instead, the virtual node structure is realised by the definition of special classes of library units(which will be described in the subsequent sections), and the formulation of a set of "composition rules" controlling their use. These rules are designed to meet the

following criteria:

- to allow the sharing of information(e.g type definitions) between virtual nodes, whilst avoiding the sharing of objects.
- to provide virtual nodes with well defined interfaces by which the entries intended to be callable by other virtual nodes can be made visible.
- to ensure that, apart from these dependencies, virtual nodes are completely independent subsystems.

5.4.3.4.1 Template and non-template units

Sharing of library units between virtual nodes is only acceptable if the units do not contain, or give access to the declaration of an "object" which would associate an internal state with the unit, such as a variable, a file , a task object or an external device. The only units which can be shared amongst virtual nodes are those which do not contain, or affect any such objects. These are called "template units" because one of their main role is to define type s which are subsequently used as a "template" for creating objects.

Template units are not allowed to reference, by way of "with" clauses, any non-template unit which might possess objects having an internal state. However, template units are allowed to "with" other template units. A unit which possess an object as defined above can only reside within one virtual node, and if it is to be indirectly used by others this might be by communication over the appropriate interface.

5.4.3.4.2 The interface units

As described before, the rendezvous has been chosen by the DIADEM project as the means of inter-virtual node communication. Therefore, the next requirement is to provide virtual nodes with well defined interfaces. This service is provided by another special category of library unit - the "interface package". such interfaces make visible the

minimum information necessary for communication in order to preserve the modularity of virtual nodes. An interface package contains one or more "interface tasks" which define the set of remotely callable entries. These represent the externally visible operations provided by the virtual node. A given virtual node may possess as many interface packages as appropriate for the communication pattern of the system.

5.4.3.4.3 The Root procedure

The final requirement is to ensure that, apart from the sharing of template units, and references to interface packages for communication purposes, virtual nodes are independent of one another.

In the case of virtual nodes, a library units must be disjoint. This ensures that no shared component exists between the virtual nodes other than the template units. This is achieved by making the rule that non-template units, which conceptually reside within just one virtual node, may only have "with clauses" naming other template units, interface packages, or non-template units residing in the same virtual node.

In addition to any part it may play in the functioning of a virtual node, the library unit representing the virtual node root has two main roles; it acts as the starting point for the virtual node's dependency graph, and must provide a means of generating a separate "thread of control" in which the virtual node executes. Amongst a number of suitable candidates(e.g. task bodies and procedures) for the role of representing virtual node root, procedures were chosen**[5]** because they are the units recognised by Ada as executable main programs, and they provide an elegant method of instantiating virtual node types for execution in the same software node. Fig.47 illustrates how the various categories of compilation units are used to build virtual nodes. This shows a typical pair of communicating virtual node objects which contain a moderate number of non-template units.

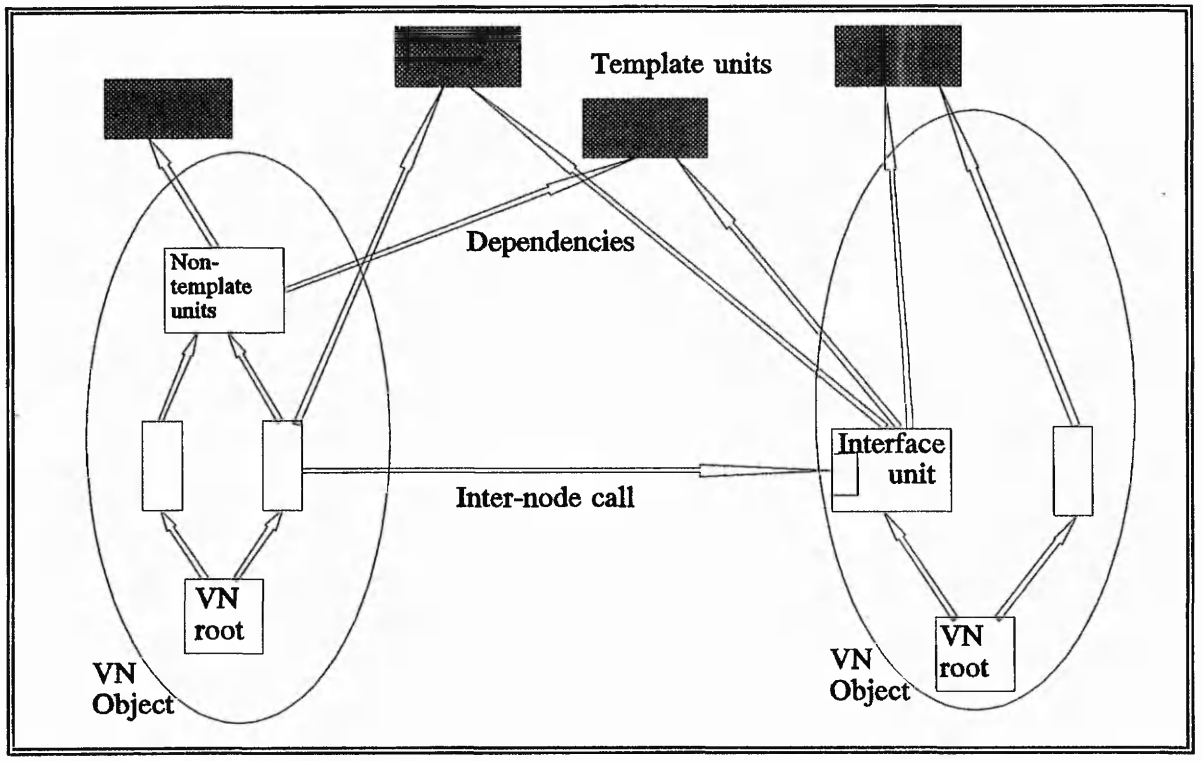

Figure 47: The Virtual Node Structure.

5.4.3.4.4 Virtual node types

On of the fundamental principles of a "type" is that each derived instance should possess all the attributes defined for the type. In particular, each instance should possess the same number of component objects. Copies of objects declared in the root procedure of a virtual node type will be created each time the procedure is called by a "thread of control" task. However, objects defined in non-template packages higher up in the virtual node dependency graph will not be recreated for each virtual node instance. This is due to the fact that an Ada virtual node requires copies of all the objects defined in the transitive closer of the virtual node roots context clause, but because of the semantics associated with main programs this does not occur. Only one copy of the objects declared outside the root procedure will be created the first time the procedure is elaborated. Consequently, multiple virtual node instances share access to a single object rather than each having a separate copy. This problem is alleviated by allowing only a single virtual node type to every node, since instantiation of the type is carried out by down-loading an

d starting up a new copy of the program. This is guaranteed to re-create all component objects and insure that all instances are identical. Unique virtual node objects are not affected by this problem either, since by definition they are never duplicated.

To provide a true representation of virtual node types instances of which may be generated dynamically and called by other virtual node, it is necessary to have at least one level indirection in the naming scheme. This allows references to a newly created instance to be bound dynamically when it comes into existence.

The only indirection facility provided by Ada is associated with the naming of instances of a task type by means of access variables. This facility is used in DIADEM project to support callable virtual node types(i.e. server types) by defining the form of the interface of a virtual node type as a task type declared in a globally visible template unit. Each new instance of the virtual node dynamically instantiates a copy of the interface task by means of an allocator function executed when the root procedure is called. The corresponding access value is then exported to potential clients to allow them to call the interface task. Effectively, these access values represent capabilities, the possession of which permits one virtual node to call another.

The provision of dynamically created interface tasks is dependent on the passing of access values between nodes of the network. For communication purposes, an access value is only used on the machine where it originated.

5.5 Conclusion

This chapter discussed the spectrum of possibilities regarding the design of distributed computing system to be used in the context of water distribution systems. One of the primary requirements of a water distribution system is the ability to expand. Expansion of the network is closely mirrored by the increased complexity of its mathematical model thus requiring more computational power. The distributed computing model presented consisted of two types - namely the tightly-coupled and the loosely-coupled systems. The most suitable type satisfying the aforementioned requirements of the water distribution

system was to be that of the loosely-coupled system.

Moreover, in order to develop application programs for loosely-coupled systems, the programming language has to have certain characteristics. These are ; software configurability, inter-process communication mechanism(synchronous or synchronous) and finally the partial failure mechanism (e.g Safety Critical systems). The suitability of each language would be measured on how many of these requirements it can satisfy. The Ada language satisfies most of the requirements, however, it is generally acknowledged that the ADA language support in the area of distributed systems is lacking.

Several strategies have been devised**[23,4,138,30,24,25,71,132,84,36,131,68,5]** to overcome the problems associated with lack of language support in this area. These strategies tend to agree that the best approach to program distributed applications in Ada, is pre-partitioning of the application program. On the subject of inter-process communication mechanism two methods predominate - namely the Remote Entry Call and Remote Procedure Call. There are strong arguments for and against using either of these methods. However, the conclusion here is that, the interprocess communication required of an application program should be the deciding factor(Hosseinzaman and Bargiela ADA-UK Conference**[61]**).

CHAPTER 6: Implementation and Design.

6.1 The program overview

The mathematical model of water distribution systems, derived in Chapter 1, is strongly nonlinear. To solve such nonlinear systems of equations iterative solution methods need to be employed (e.g. Hardy Cross, Newton Raphson ...etc). From amongst the methods studied in Chapter 1 Newton Raphson method was selected to solve water distribution system simulation problems.

However, the major drawback of the water system simulation program, incorporating an iterative solution method, was its inability in dealing with rapid growth in the computational requirements with increase of the physical network size. A classical way of overcoming this problem is to partition the system into smaller subsystems, solve the derived subsystems then combine the subsystem solutions to yield the overall network solution. This approach lead to the development of a nonlinear network tearing algorithm (described in Chapter 3), which is derived from the original work done by Kron**[80]** on tearing linear systems. The nonlinear network tearing algorithm introduced a particularly efficient method of calculating a coordinated solution as it is a direct analytical technique avoiding any iterative recalculation of subsystem solutions.

The water distribution system is thus solved by consecutive Newton_Raphson iterations interleaved by diakoptical coordinated solution to linearized subsystems representing iterative corrections.

6.1.1 System's functional units identification and data flow

In the water system simulation program(Fig.48(a)) the topological data input contains the erroneous estimates of the system's pressure measurements, this data is transformed by the program to give the true pressure estimates of the system. The program is comprised of state estimator program incorporating a network tearing algorithm(Fig.48(b)). The water distribution system pressure estimates are derived by a sequence of Newton

Raphson iterations with each iterative correction to the pressure estimates calculated as a diakoptical coordinated solution to linearized subsystems.

With reference to the mathematical model of the water networks the main computational effort involved in solving equations (84) and (85) is the inversion of the Jacobian matrix J. Applying nonlinear diakoptics to the system results in the partitioning of J into blocks representing the jacobian of the subsystems. Solving (84) "block-at-a-time" and then coordinating the partial solutions(Fig.48(c)), computational saving can be achieved. However, while the "partitioned jacobian" approach works well when implemented on a single CPU computer it is not best suited for parallel distributed processing architecture as it tends to overburden the coordinating task with additional calculations and sending large amount of data associated with each jacobian block. Furthermore, sending large data in a distributed system creates large overheads which ultimately result in large processing times. Consequently the parallel processing gains are easily wasted on coordination and communication overheads. To avoid such problems the network is partitioned before its linearization. This means that subsystem jacobians are calculated on the individual processing units(Fig.49). Thus, only a small amount of data is transferred between the coordinating unit and the worker units.

$$\Delta x = J^{-1}(Z - g(x^k)) \qquad (84)$$

$$x^{k+1} = x^k + \Delta x \qquad (85)$$

The design concept used to define the distributable components of the water system simulation program is that of "virtual node". Consequently, the preferred method communication**[5]** is the remote rendezvous mechanism. The remote rendezvous end-to-end protocol consists of a layer of software appropriately called Remote Rendezvous layer. The main components of this layer are illustrated in Fig.50. Each Ada node which can serve as a callee in a transaction has a permanent task which listens at a prearranged communication port for incoming messages signalling an entry call. When this so called entry port task has determined which entry the call is intended for, it

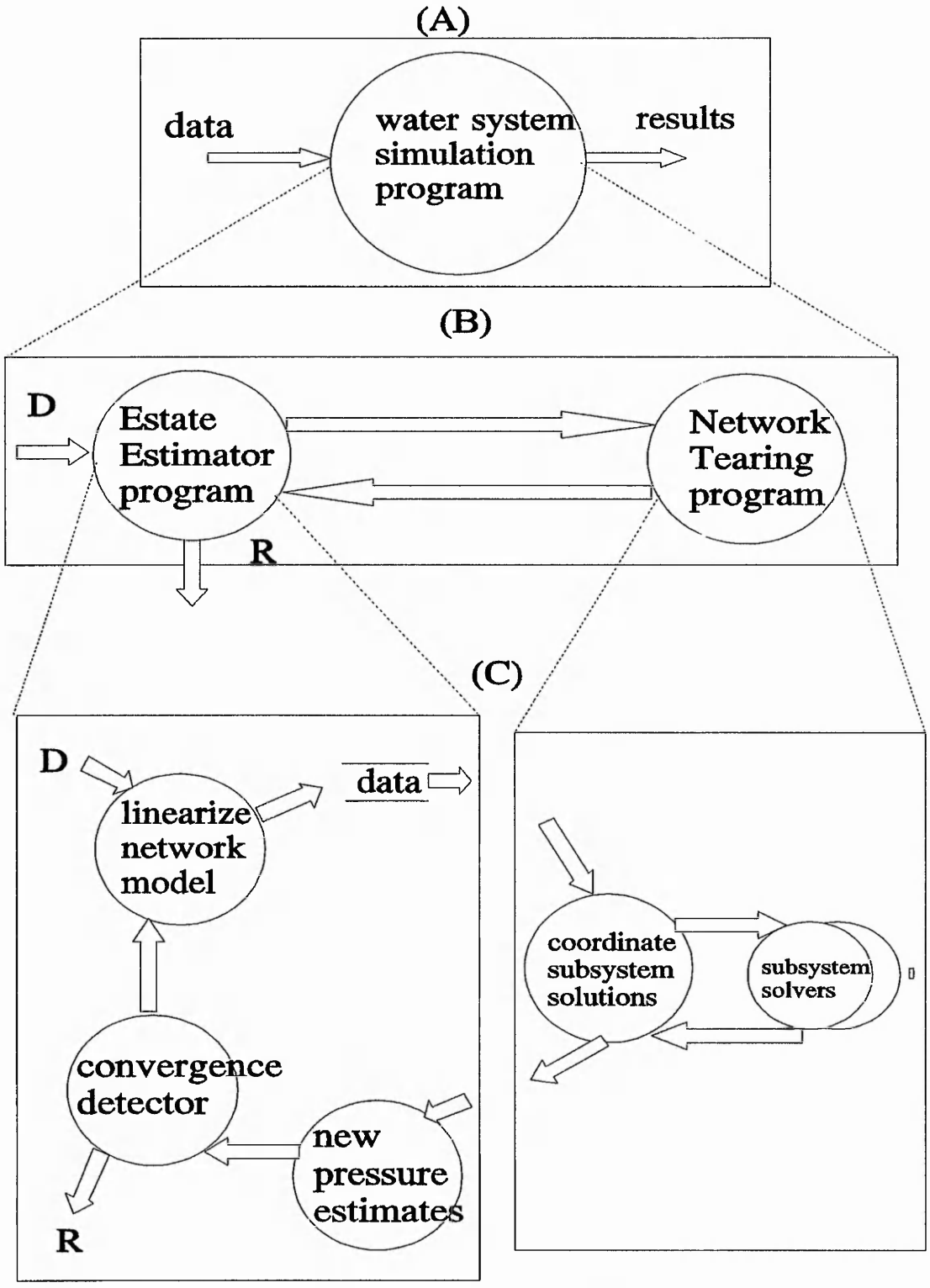

Figure 48: The data flow diagram for Water system simulation program.

generates a local agent task to issue the call on behalf of the remote caller. The called task therefore experiences precisely the same call it would have experienced had the call not been remote. On completion or failure of the call, the local agent task communicates the result directly to the caller and then terminates. All the components making up the remote rendezvous layer must interact with the network communication software to transfer messages between the various physical nodes. Referencing the ISO model, on the other hand, the lowest layer that provides end-to-end service is the transport layer, thus the remote rendezvous layer must interact with the transport layer of the underlying communication system. However, there are differences between the transport layers of different systems, thus in order to harmonise these differences another layer of software is introduced between the remote rendezvous layer and the communication system. This layer takes the form of a predefined package called STANDARD_COMMS(described before). This package provides the message passing primitives required to support remote rendezvous transactions. In this scheme software in a distributed system is thought as consisting of the following layers(Fig.51): (i) Application software layer, (ii) The remote rendezvous layer, (iii) The standardising layer, and (iv) Host communication/operating system layer.

6.1.2 Design of functional units

The task of programming the water system simulation algorithm in Ada using the "virtual node" concept is performed in two stages: first the algorithm is segmented along the functional boundaries and defined as virtual nodes, and then virtual nodes are transformed into what is called "Ada nodes". The transformation mainly involves the introduction of extra code to support remote rendezvous transactions. In order for the Ada virtual nodes to communicate with other virtual nodes in the network, they are given an interface package. The interface packages provide the remotely visible entries, defined within the interface task that can be called by other Ada nodes.

This research resulted in an efficient implementation of Ada Virtual Nodes.

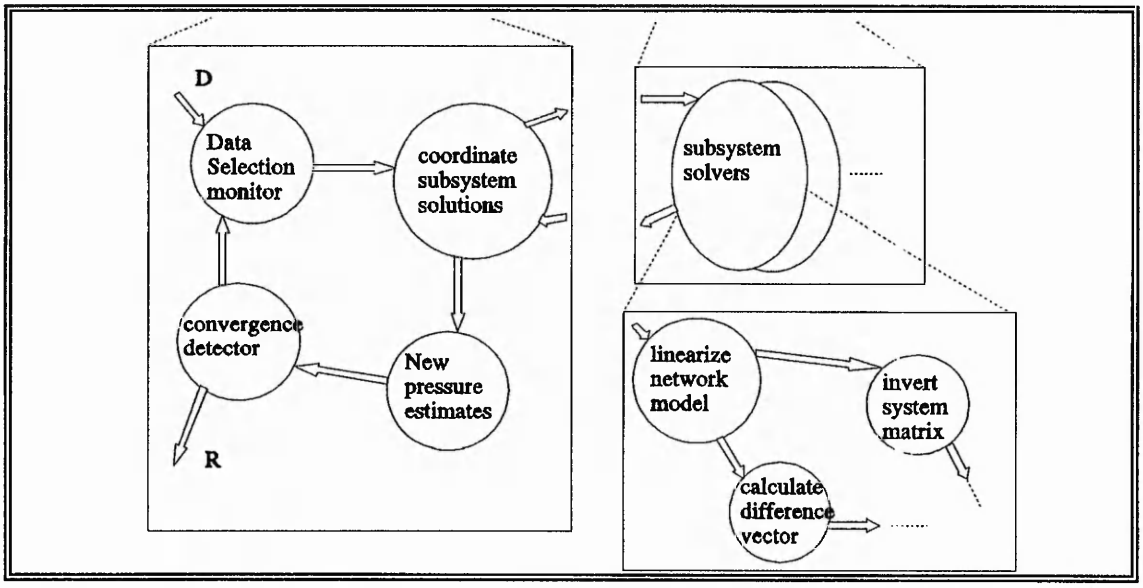

Figure 49: The new subsystem solution routine calculating the subsystem jacobians locally on the individual processing units.

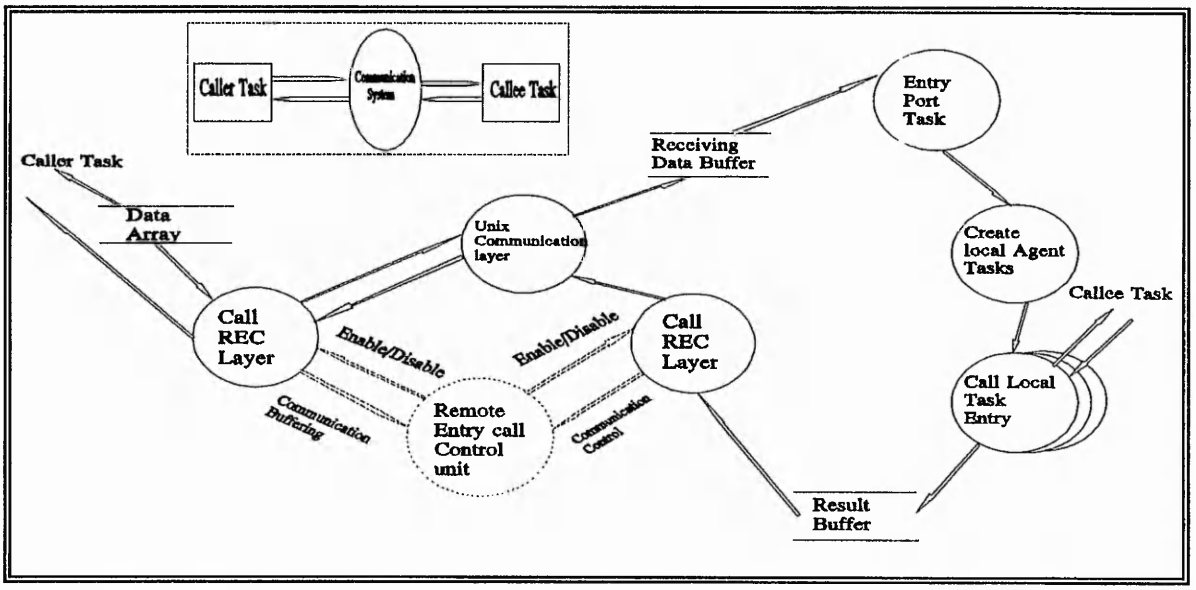

Figure 50: The Remote Rendezvous Mechanism.

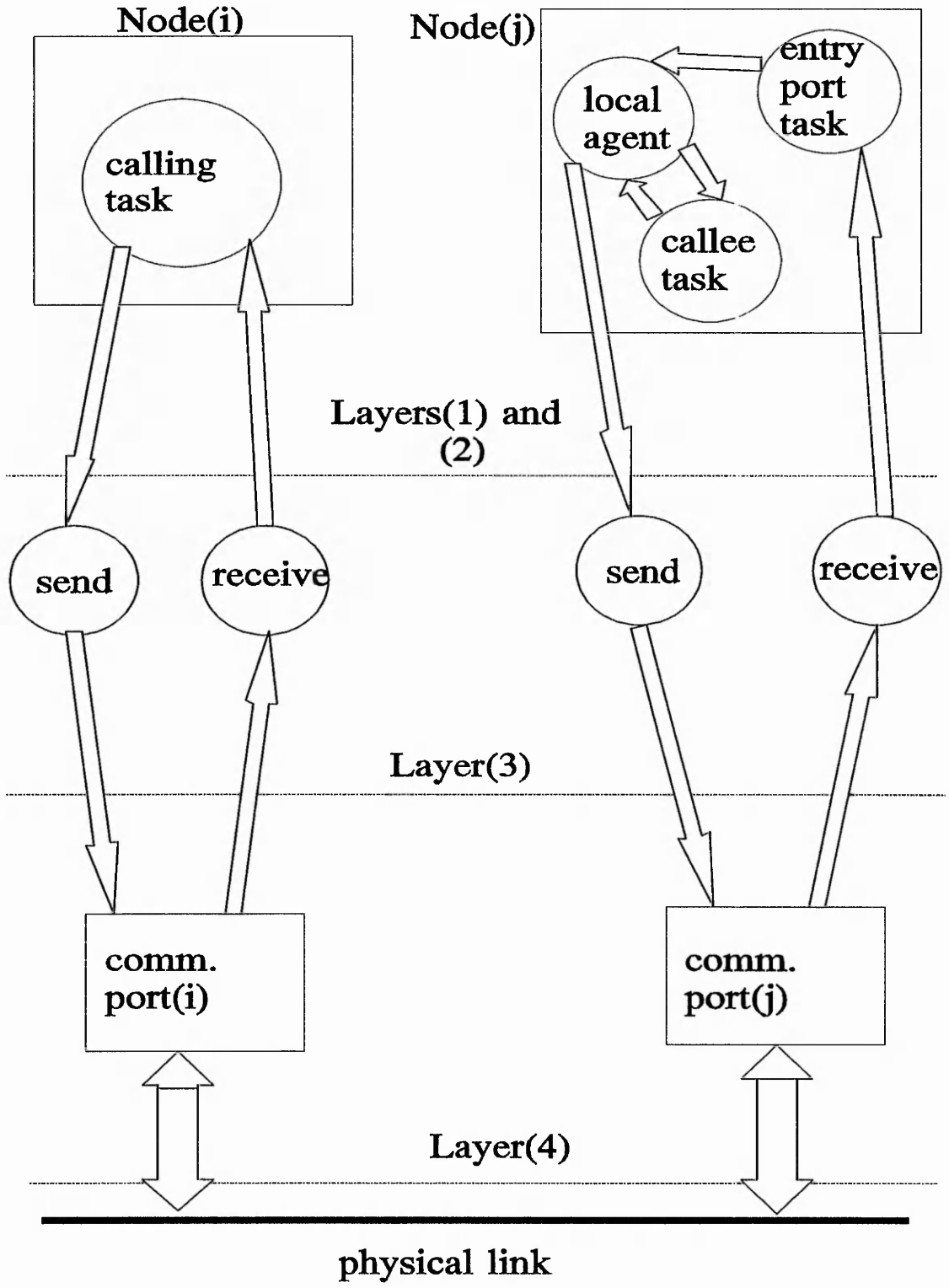

Figure 51: The data flow in a CLIENT/SERVER communication system.

In the water network simulation program, it was necessary to identify the program components that can be programmed as virtual nodes or virtual node types. On the other hand, the decomposed model of the water distribution system, comprised of coordination and subsystem solution processes, which can be programmed as independent virtual nodes. The subsystem solution processes do vary in numbers according to the partitioning scheme employed, thus they can be coded as virtual node types which can be instantiated dynamically. The process of coordination, on the other hand, is the same for every partitioning scheme, thus it should be regarded as a unique virtual node which is generated statically.

The coordinating program (Fig.52) initially receives the worker interface task's addresses (including their access values) as they are instantiated by their root procedures. This would enable the coordination program to call the target worker's interface task. It is acknowledged that the practice of sending access values to another machine in the network is hazardous since the use of such parameter in a different address space may have unpredictable effect. However, this value is purely used for task identification purposes by the coordinating program. Having the access value of the worker interface task enables the coordinating program to send the data to the target worker, by identifying the correct instance of the worker task.

The coordination program responsibilities therefore are: first of all to prepare data destined for worker tasks. The data is encapsulated in a packet incorporating the unique identification information of each work task. Then it waits for the result packets to arrive. The order in which the result packets arrive in the coordination program's buffer is arbitrary, thus incorporated in each result is a worker identification number which helps the unpacking and storage of the result packets in the correct order. A simplified diagram of the coordination program is shown in Fig.53.

In contrast, the worker tasks send their interface task's access values to the coordination

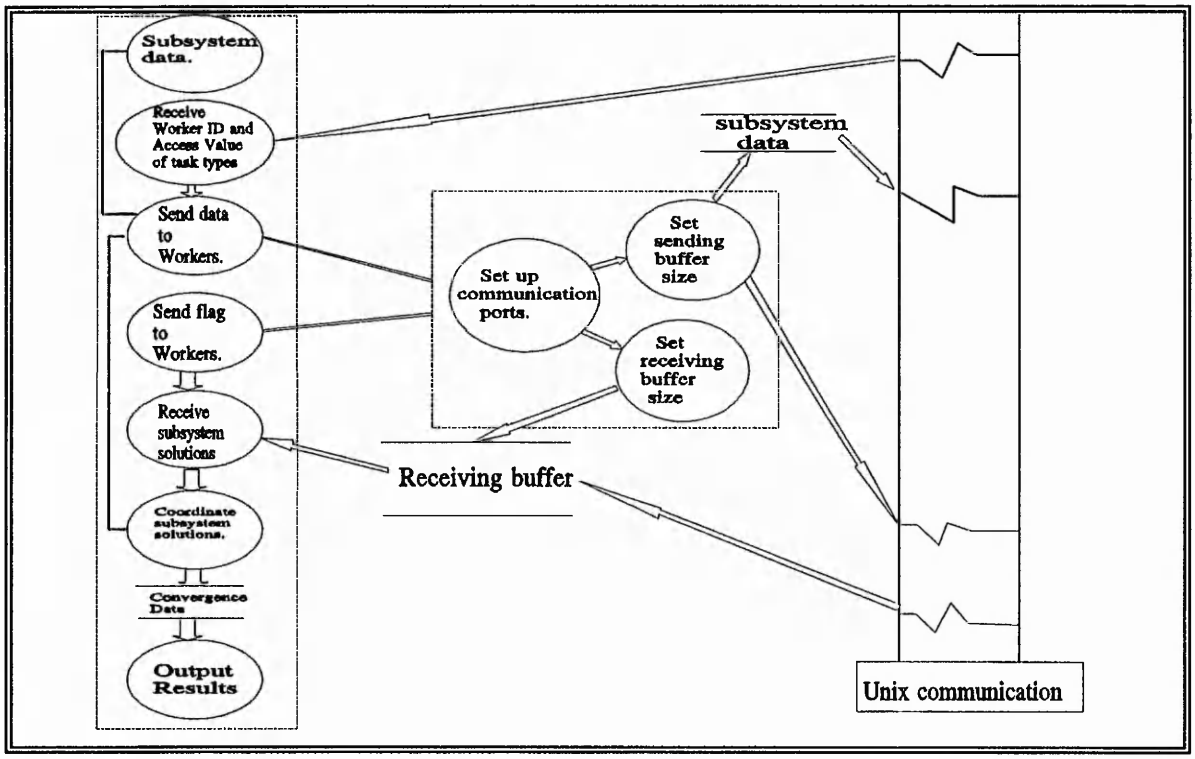

Figure 52: The Coordination routine components.

program, following their dynamic instantiation. The worker task then waits for the incoming data packets dispatched by the coordination routine. Upon receiving the data packets, the access value of the worker interface task, unpacked from the data packet, is used by the worker task in order to call the interface task of the target worker. This ensures that the correct instance of the worker virtual node type receives the data. Initially the data arrives in the "entry port task". This task's responsibility is to monitor the communication ports for incoming data. Subsequently, the entry port task generates a dynamic "local agent task" (i.e SOLVER_TASK_AGENT) and passes to it the actual parameters of the remote call. A simplified diagram of the subsystem worker tasks is shown in Fig.54. The entry port task then loops back to repeat the process for other calls. The actual entry call is made by the surrogate SOLVER_TASK_AGENT in the form of a local entry call. When the entry call is accepted and the call is completed, the worker program waits for an acknowledgement from the coordination program to send the result packets.

6.1.3 Virtual Node Design Process

With reference to the preceding section, it was decided to program the coordination process as a unique virtual node object and the subsystem solution process coded as virtual node types. since the coordination process is a unique component that exists throughout the life of the system, it is clearly best modelled as a unique virtual node object. The number worker tasks performing the subsystem solutions however, varies dynamically to suit different partitioning schemes. Consequently, these are best implemented as virtual node types which can be instantiated dynamically. Fig.55 illustrates the water system simulation program's Virtual node structure.

Interface units

Unlike the interface tasks of virtual node types those of unique virtual node objects are statically defined in interface packages. Considering our water system simulation program, the coordination virtual node for example has one entry, REQUEST_INITIAL_DATA, for receiving the access values of the worker tasks and another entry, RECEIVE_FINAL_RESULTS, for receiving the final results(i.e. subsystem solutions).The concept of a type, on the other hand, requires that each derived instance should structurally, and functionally be identical. In particular, each instance should have the same number of objects. Copies of objects declared in the root procedure of a virtual node type will be created each time this procedure is called by a thread of control task. However, objects defined in non-template packages will not be recreated for each virtual node instance. Only one copy of these objects is created the first time the root procedure is elaborated. To overcome this problem an Ada node type is derived from a single virtual node type, since instantiation of the type is carried out by down-loading and starting up a new copy of the program.

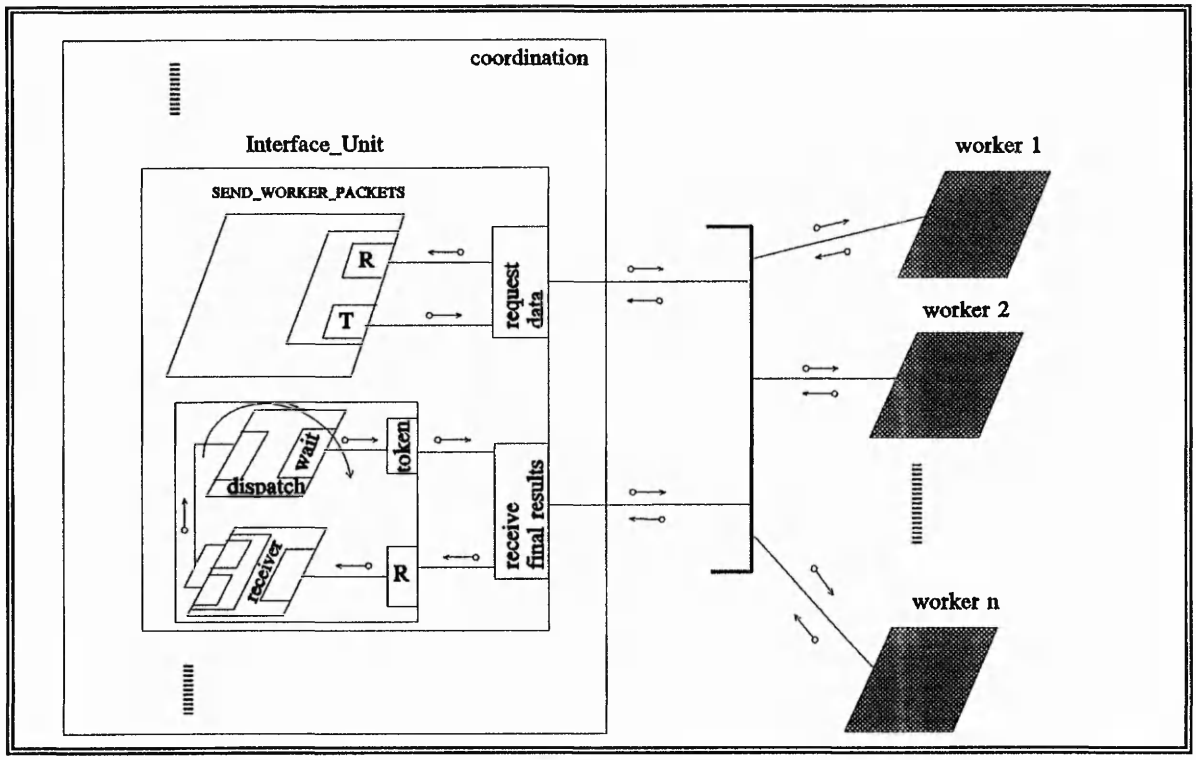

Figure 53: Simplified diagram of the coordination routine.

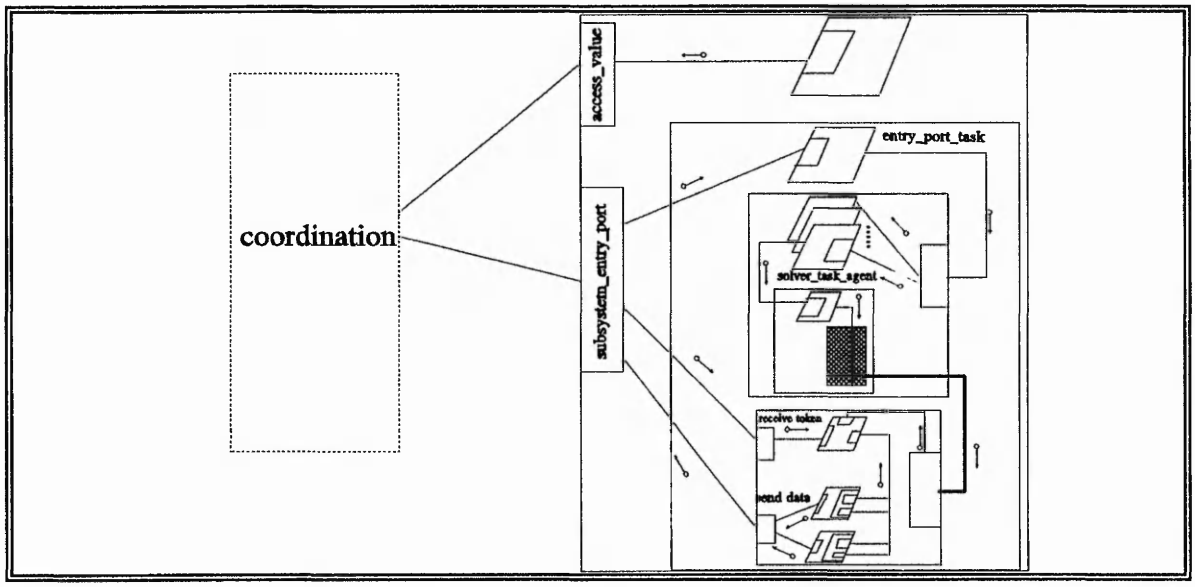

Figure 54: Simplified diagram of the subsystem worker task.

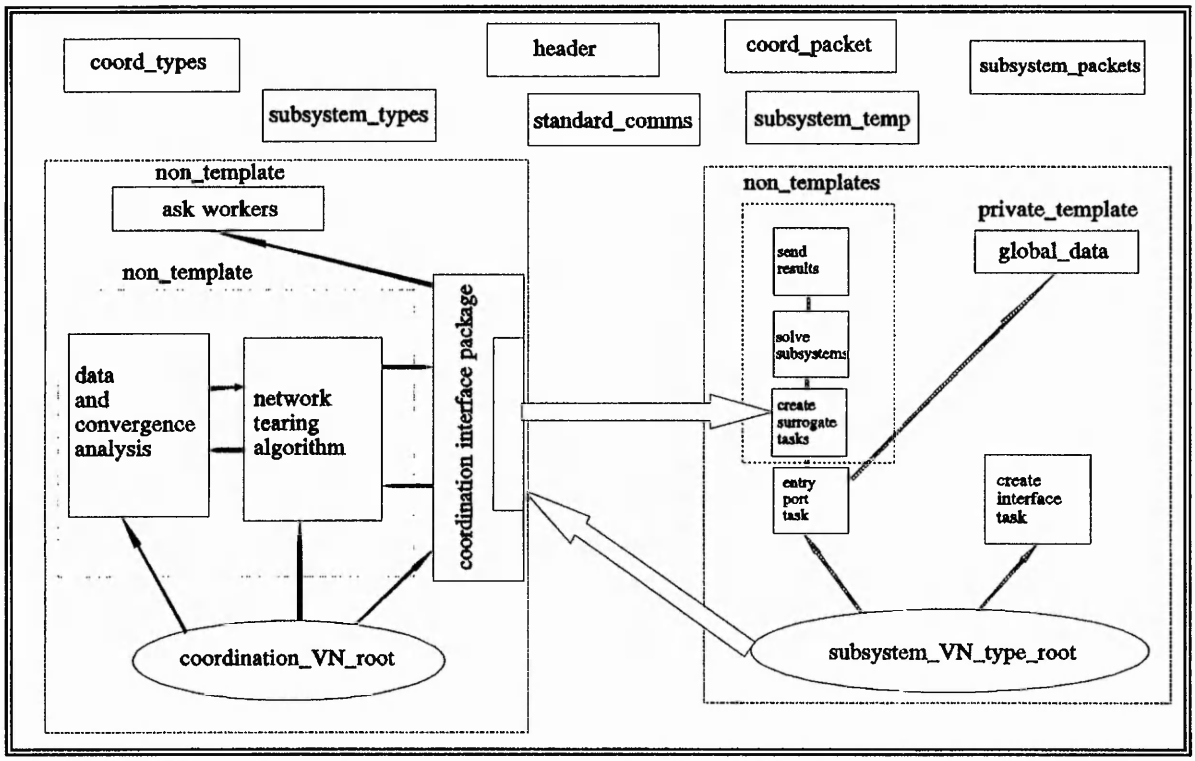

Figure 55: The water system simulation program Virtual Node structure.

To provide a true representation of a type, instance of which may be called by other virtual nodes, it is necessary to have a level of indirection in the naming scheme. Ada provides such an indirection facility for task types by means of a "task access value". To support an equivalent facility for virtual node types it is necessary to allow access values to be passed between virtual nodes resident on different machines. This is precisely the reason for the existence of module "SEND_ACCESS_VALUE" in Fig.54. The root procedure of the "SUBSYSTEM_VN_TYPE" dynamically creates an instance of a globally visible task type. This is possible since the task type is defined within a template unit. It then makes a call to the coordination virtual node and sends it the access value of this dynamic interface task. The coordination virtual node therefore registers each particular instance of the subsystem virtual node type as soon as it is brought into existence. The coordination virtual node thus has a collection of access values which

record all current instances of the task types, and uses them to call the particular subsystem virtual node type.

Template and non-template units

The definition for COORDINATION_PACKETS is used by both coordination virtual node and subsystem virtual node types, therefore it is defined as a template package. SUBSYSTEM_PACKETS and SUBSYSTEM_TEMPLATE are referenced by them for inter-virtual node communication, thus they qualify as template units. Both the COORDINATION_TYPES and SUBSYSTEM_TYPES are referenced by the template packages above, thus they may reside within the template library. The two special units, HEADER and STANDARD_COMMS, may also reside within the template library since they are referenced by the coordination and subsystem virtual node and virtual node types respectively. Moreover, they only define the templates for the communication and interfaces to the underlying operating system (which will be explained in the next section).

In Fig.55 units identified as non-template, for both coordination and subsystem virtual node and virtual node type respectively, do not satisfy the composition rules. On the other hand, the PRIVATE_TEMPLATE units represent units that are only used by the subsystem virtual node type. However, this does not imply that unique virtual nodes such as the coordination program can not have their own private-template units. The components within either the coordination virtual node or subsystem virtual node types can gain access to the template unit by the use of a "with" clause.

6.2 Implementation details

The process of programming the water system simulation program in Ada consists of two phases, first the development of programming model for distributed environment(Ada "virtual node" approach) and then the construction of a remote rendezvous layer for inter-process communication. The preceding sections have comprehensively covered the

programming model, hence the virtual node approach. In particular the data flow analysis allowed identification of the system's functional activities, and the communication pattern between these entities. In addition, it identified that the distributable elements can be suitably represented if they were partitioned into two classes, namely the coordination process and the subsystem solution process. Using this model for developing the virtual nodes resulted in the creation of a coordination unique virtual node and subsystem virtual node types. However, detailed design of these entities is found in the preceding sections and would not be elaborated further here. The remaining parts of this section will therefore be concentrating on issues that were not covered by the previous sections.

6.2.1 Inter-virtual node communication issues

6.2.1.2 Dynamic communication port creation

The first requirement of the standard interface discussed in the preceding sections, is that of the communication layer, upon which remote rendezvous layer is based, must support the dynamic creation of ports to act as communication end-points. The explicit use of primitives(e.g. OPEN and CLOSE) for dynamic creation of communication ports is discouraged if the communication end-points provided by different environment are to be used. Thus, there can not necessarily be a correspondence between the "logical ports" viewed by the remote rendezvous layer, and the physical communication end-points created in the underlying communication system.

To overcome this problem, the required communication end-points are created in the initialization part of a generic package. These logical ports can only be accessed the associated communication primitives defined in the package. In our implementation the operating system in use is UNIX, the communication end-points are called "sockets". In order to create the sockets dynamically, two separate generic package are defined in the standard interface, namely the CALLER_PORTS and the CALLEE_PORTS. These generic packages provide the communication primitives. For instance, if a caller wishes to call another unit on the callee side, the procedure SEND_CALLER, defined in the

generic package CALLER_PORTS of the package STANDARD_COMMS along with other primitives(e.g. RECEIVE_CALLER procedure), is called. The body of the SEND_CALLER instantiates the generic package CREATE_COMM_CALLER_PORTS, defined within HEADER package, which creates the UNIX socket communication end-points. Similarly, there exists procedure SEND_CALLEE for calling units on the caller side.

6.2.1.2 Network wide ID for distributed components

The usage of generic packages in the process of creating the communication end-points automatically associates, by the naming rules of Ada, the communication primitives with the encapsulating package. In fact, the name of the package can be regarded as the name of the pair of logical ports and appears in the full name of the associated primitives.

However, high-level naming of ports are unavoidable, since in order to issue a remote entry call, a task must explicitly identify the static port of the node it wishes to call. In our program the generic packages, whose instantiation creates the communication ports, retain the addresses of both the caller and the callee nodes. Thus instantiating these packages would automatically assign to them their physical node ID and their port addresses. When both the caller and the callee ports are created, all that is needed by the caller to call the callee task is only one system independent identifier - namely the software node identifier. This would assign a unique network ID to the caller port, which is sent to the callee node as part of the original message. Therefore, the network name table store: the software node ID, physical node ID and the port-address for each software node in the system. The NFS facility of UNIX operating system, on which our implementation of the remote rendezvous is based, provides the network wide visibility of files. Thus, system wide name management can be based on a common file stored on a single node. This has the advantage that new software nodes can be easily added to the system at any stage simply by updating the common file.

6.2.1.3 Marshalling and unmarshalling of data

In the remote rendezvous call, messages arriving at the callee's receiving port would be of different sizes and types, this can create a problem since it would be almost impossible to predetermine in which order they may arrive. To alleviate the problem, message packets are transmitted as "records", this provides a convenient way of encapsulating different data types. These records consist of different fields that can contain different types of data. In fact, record types define a contiguous section of memory within which memory fields are reserved for specific objects. Thus they provide a suitable constructs for messages transmitted over the network. However, although records are meaningful to Ada, but they are regarded by the communication layer as strings of bits on both ends of the communication network. Therefore to allow conversion of the received data to the required type, data packets contain information to that effect.

The remaining problem would be, how to convert the received data to the required type?. One way would be to use the predefined package UNCHECKED_CONVERSION. The use of such package for type conversion is not recommended by Ada RM**[1]** unless in extreme circumstances. The alternative approach adopted by DIADEM is based on the use of variant records. All incoming messages to a callee node are variants of a single record type covering all the remotely visible entries, the discriminant being an enumeration type for the target entry. The advantage of this approach is that the type conversion is done automatically when a message is received, and requires no explicit type conversion.

6.2.1.4 Message buffering and flow control

The preceding section introduced the notion of variant record types representing the data packets for transmission over the network. The space for the record is allocated in the normal way by the Ada runtime system. However, in order for messages to be sent or received from a communication port, they must reside in the memory associated with the port. In this case, it is achieved by simply copying the data from the record allocated by the Ada runtime system to the space allocated to the communication ports.

During a remote rendezvous transaction, the space for receiving incoming messages is created by the communication system and released by the Ada program after use. Conversely, the space for sending outgoing messages is created by the Ada program, and released by the communication system once the message has been dispatched. In order to hide the specific technique employed to perform these operations, two special primitives are provided. The ALLOCATE procedure returns an access value to a record of type CALL_PACKET, referenced by the REF_CALL_PACKET, generated by the communication system. The DEALLOCATE procedure accepts an access value to a record of type ANSWER_PACKET, referenced by REF_ANSWER_PACKET, and releases the space.

The inter-process communication facilities provided by UNIX is the connectionless datagram protocol(UDP), as mentioned before. This protocol does not provide any flow control facility, thus messages sent by different workers simultaneously can be lost or corrupted. To prevent situation, a form of handshaking is employed to synchronize the transmission. The workers do not send data unless they receive a signal from the callee side indicating that the callee virtual node is ready to receive data from the caller virtual node. The flow control mechanism described here is shown in Fig.56.

6.2.2 Multi-library mechanism

The distributable virtual nodes and instances of virtual node types are organised into two libraries - namely the "coordination component library" and the "subsystem worker library".

These libraries were created using the Alsys multi-library environment tool**[2]**. In the multi-library environment, the libraries are organised into families. The libraries of a family can import units from each other. This is done through the link mechanism, which effectively creates a reference from the importing library to the original library that holds the unit. The object code, symbol tables and other information associated with the actual compilation unit itself remains in the original library.

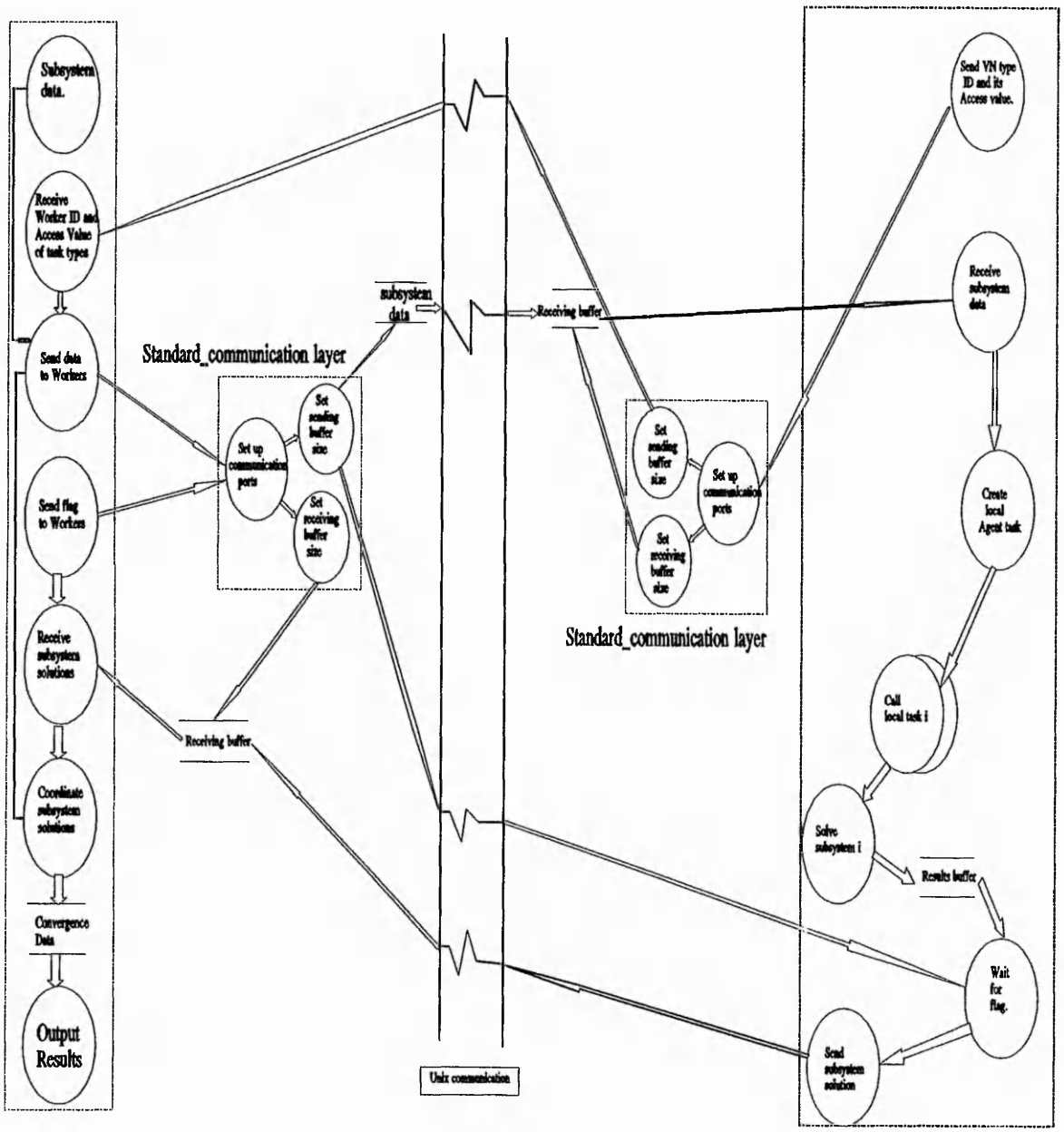

Figure 56: The structure of distributed Water system simulation program.

For example, considering Fig.57, apart from aforementioned library units(i.e. coordination and subsystem worker libraries) exists another library which houses those components that are shareable between the virtual nodes and instance of virtual node types - namely the "template" library. Using the link facility, components of both coordination library and subsystem library can reference the template units situated in the template library but part of the same families of libraries.

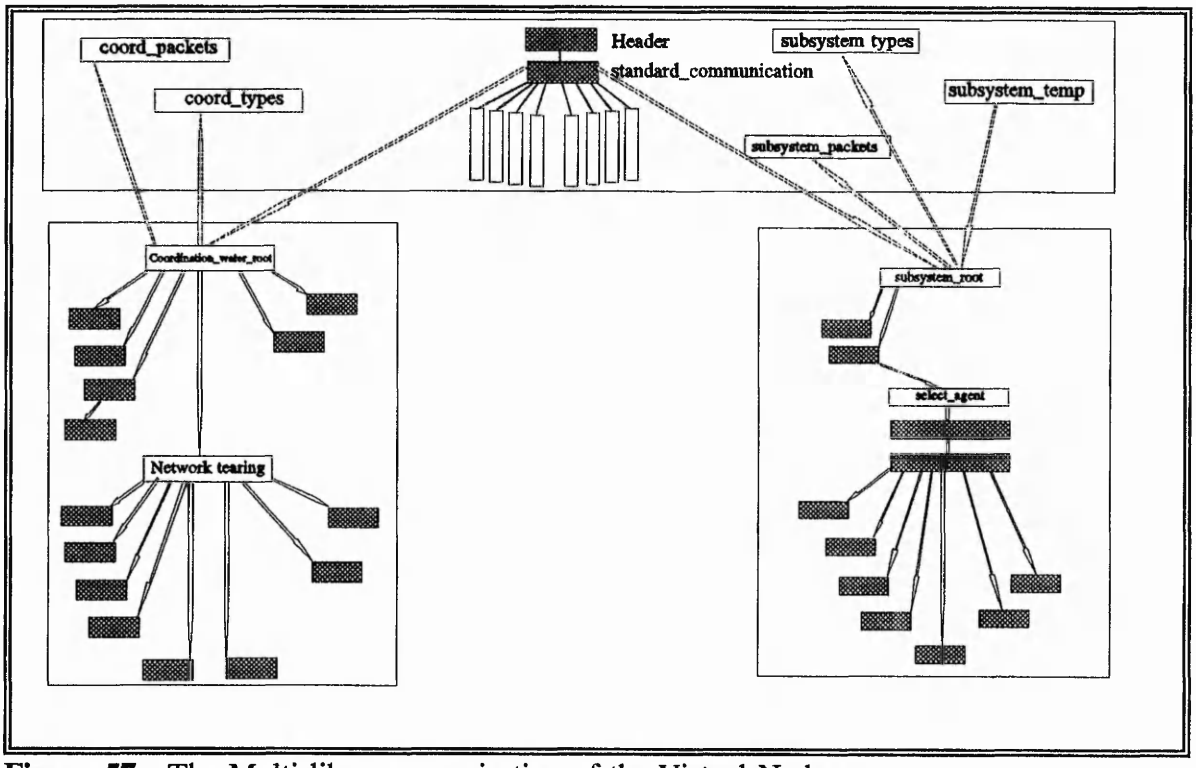

Figure 57: The Multi-library organisation of the Virtual Nodes.

The specification of the interface task of the worker's virtual node types is defined within a template package as was explained in the preceding sections. However, the body of the interface could not be defined within the same template package since this would have compromised the composition rules. To overcome this problem, the interface task specification had to be copied to the subsystem worker library. The copying of such units would mean that all the units which the template unit is dependent on, had to be copied as well.

Clearly this would have defeated the idea of multi-library environment. Fortunately, Alsys multi-library mechanism provides an alternative to straight forward copying of files, this facility is known as "Alternate version of bodies", This mechanism allow two compilation units to share the same actual parent unit while they exist in different libraries. Thus, it was possible to define the specification of the interface task of the virtual node types to exist in the template library while having a body defined in subsystem worker library.

6.3 Performance and Results

The primary objective of the final part of this report was to explore the possibility of developing a suitable environment for the implementation of our nonlinear network tearing algorithm**[60]** in a loosely-coupled computing network, using the Ada language. Moreover, the computational performance of the algorithm would be examined, and the results shall be examined to determine the suitability of such an environment.

The performance of the water distribution system simulation program was evaluated using data obtained from two realistic pipe networks - the 65 nodes and 130 nodes networks. The algorithm was programmed in Ada using the "virtual node" concept to develop its distributable components.

The program is implemented on a network of SUN/SPARC workstations connected by an Ethernet communication link, having a UNIX operating system. The inter-process communication provided by UNIX is based on connectionless User Datagram Protocol(UDP) sockets.

6.3.1 The Program

The structure of the simulation program incorporating a nonlinear network tearing algorithm is shown in Fig.56. The decomposed mathematical model of the water distribution system is comprised of a coordination routine and a subsystem solution

routine(as was mentioned before). These entities resembling the distributable components of the simulation program, were coded as a unique virtual node object and instances of a virtual node type respectively. The communication mechanism in use, is a remote rendezvous mechanism which is an extended version of the standard rendezvous system employed in the Ada language for inter-task communication. The extension is necessary to cater for the remoteness of the processes. The communication pattern is based on a CLIENT/SERVER model. The remote rendezvous mechanism is built on a connectionless Internet user datagram protocol.

The coordination virtual node(VN) is started before the subsystem program, since the coordination VN represent the SERVER node in the implementation and all the instances of subsystem VN types are its CLIENTs. The coordination VN begins its task by reading the system data. This data is then segmented into required number of packets, available for transmission to the subsystem VN types. As was mentioned before, for every callable VN in the system there is a package defining the "call" and "answer" message types for each remotely visible entry. In the case of coordination VN, this package is known as COORDINATION_PACKETS and packets are defined as record types - the CALL_PACKETS and the ANSWER_PACKET.

However, the data segmentation is suspended until after the reception of a message from each of the subsystem VN types. These messages represent the access values of the interface task of the instances of the subsystem VN types. This value together with other identification parameters, incorporated into the data packet helps identifying the correct instance of the subsystem VN types.

Segmentation of the data resumes and the partitioned data is stored in the dynamically allocated data packet(ANSWER_PACKET in this case) of the COORDINATION_PACKETS package. Before sending the packet, it is important to determine its size(in bytes), since this value is required to inform the underlying communication of the size of the packet being sent. Furthermore, system calls initiated by the communication layer to call the underlying communication system's primitives(e.g.

SENDTO of UDP) require the data packet to be sent by reference. Thus the constructed data packet is in fact a pointer to the data packet record(i.e. REF_ANSWER_PACKET). Once the data is dispatched to the individual subsystem worker VN types, the data packet is deallocated.

The entry port task of the subsystem VN types detecting that data is arrived at the communication port, allocates a data packet of the appropriate type(i.e. COORDINATION_ANSWER_PACKET in this case), to copy the data from the communication area into the Ada program memory area. The data packet is deallocated and the memory released after the data is copied.

To call the appropriate instance of the subsystem VN type, a surrogate task is instantiated to make the entry call on behalf of the coordination VN. The surrogate task is an instance of a task type known as SOLVER_TASK_AGENT, defined in procedure SELECT_AGENT. The entry call is accepted in the body of the interface task of the subsystem VN type, and rendezvous commences. The subsequent stages of the subsystem worker routine is concerned with obtaining the solution of the subsystem units. When these results are obtained, the subsystem VN types wait their turn before transmitting the result packets. This mechanism for controlling the data flow had to be introduced, at this stage, since this facility was not provided by the Internet UDP layer. This means that without it the subsystem worker VN types had no way of knowing whether the buffer on the receiving end is full or empty, thus messages could get corrupted or lost as a result. Upon receiving confirmation flag, the subsystem VN type dynamically creates the appropriate data packets(REF_CAL_PACKET pointer to the CALL_PACKET record defined in SUBSYSTEM_PACKETS package). Size of each packet is evaluated and sent to the underlying communication system's primitive(e.g. SENDTO). The dynamically instantiated packet are deallocated at this stage.

On the coordinating side, data packets arriving consecutively are collated and the subsequent stages of the coordination process ensue. The data transmission pattern, as described above, is repeated until overall system solution is obtained. The final solution

data are stored and the coordination VN and instances of subsystem VN types are terminated.

6.3.2 Discussion of Results

The initial tests were performed using data from a 65 nodes pipe network. The network was partitioned into 2 to 5 subnetworks with varying number of cut-branches. The result of these tests are presented in Tables 7(a) - 7(d), while Table 8 presents a summary of the results by considering the average value of subsystem solution time for each partitioning scheme (i.e. partition of 65 nodes network into 2 to 5 subnetworks). Similar to the transputer implementation of the algorithm**[60]** attempts were made to partition the network in such a way that resulted in subsystems of approximately the same size. The Alsys multi-library mechanism helped to organise the compilation units into independent libraries. The construction of the virtual nodes and virtual node types were finally completed using the "link" and "Alternate version" facilities of the multi-library mechanism. The link facility established the links for referencing the template units. The alternate version of the body of the subsystem interface task was created, thus hiding of implementation details of the interface task(adhering to composition rules).

The results of the aforementioned tests are presented graphically in Fig.58 and Fig.59, similar to the transputer implementation, these results reflect the time taken to coordinate subsystem solutions and the subsystem solution times respectively. The coordination time (Fig.58) includes the setup and transmission times of data packets intended for every subsystem solver executing in a virtual node.

In Fig.58 coordination times are evaluated for different partitioning scheme mentioned above(2 to 5 subsystems). The quadratic dependence of the coordination time on problem-size (i.e. number of cut-branches plus the number of subsystems derived from the partitioning scheme) is depicted, similar to the transputer implementation. This seems to reaffirm the fact that there is a point beyond which partitioning of the network into a greater number of subnetworks is not efficient. This is governed by the increase in the

problem-size, which results in the coordination time increase by increasing the coordination matrix size (i.e. representing the compensating flows in the torn networks), thus outweighing the decrease in subsystem solution times. On the other hand, the pseudo-linearity of Fig.59 indicates that the subsystem solution times increase quasi-linearly with respect to subsystem size.

The presented subsystem solution times together with the coordination times for the given partitioning schemes (65 nodes network partitioned into 2-5 subnetworks respectively) seems to indicate that, while it is advantageous to solve the subsystems concurrently in the distributed environment (i.e. network of SUN/SPARC workstations interconnected by an Ethernet communication link), subdividing the network into too many small subnetworks results in an increase in the coordination time thus outweighing the concurrent processing time gains.

The results obtained here concerns only the coordination and subsystem solution times, these include the communication overheads which may be significant in comparison with the actual time spent on coordination and subsystem solution. In our implementation, for example, the underlying communication is consisted of connectionless internet user datagram protocols(UDP), this protocol layer rests on top of the IP layer of UNIX communication system. The maximum allowable packet size by the network, including the header, is 1500 bytes(MTU of Ethernet network) which means data packets larger then 15kbytes are subject to fragmentation at the sending end and reassembly at the receiving end. This happens despite the fact that maximum IP datagram size is 65kbytes. Consequently fragmentation and reassembly times are added to the total transmission time of the data packets sent over the network**[141]**.

However, such large packets would only be sent if the network is partitioned into small number of subsystems, which does not typify a normal partitioning criteria for large networks. Two predefined packages providing system dependent features were extensively used throughout our implementation, they were the SYSTEM and CALENDAR packages. The package SYSTEM provides system dependent features of a particular computing

system. For example the system calls to the underlying communication system required the addresses of communication ports which were of type ADDRESS defined in package SYSTEM. The STORAGE_SIZE attribute defining the number of bits in each addressable storage unit of the host machine, is also defined in package SYSTEM.

The package CALENDAR, on the other hand, provides a range of procedures and functions for time and date evaluation. In particular function "CLOCK" was used, this function returns the current value of TIME at the point it is called. The value returned by this function is expressed in seconds. This function was used in order to determine the coordination and subsystem solution times of coordination VN program and subsystem VN type program respectively.

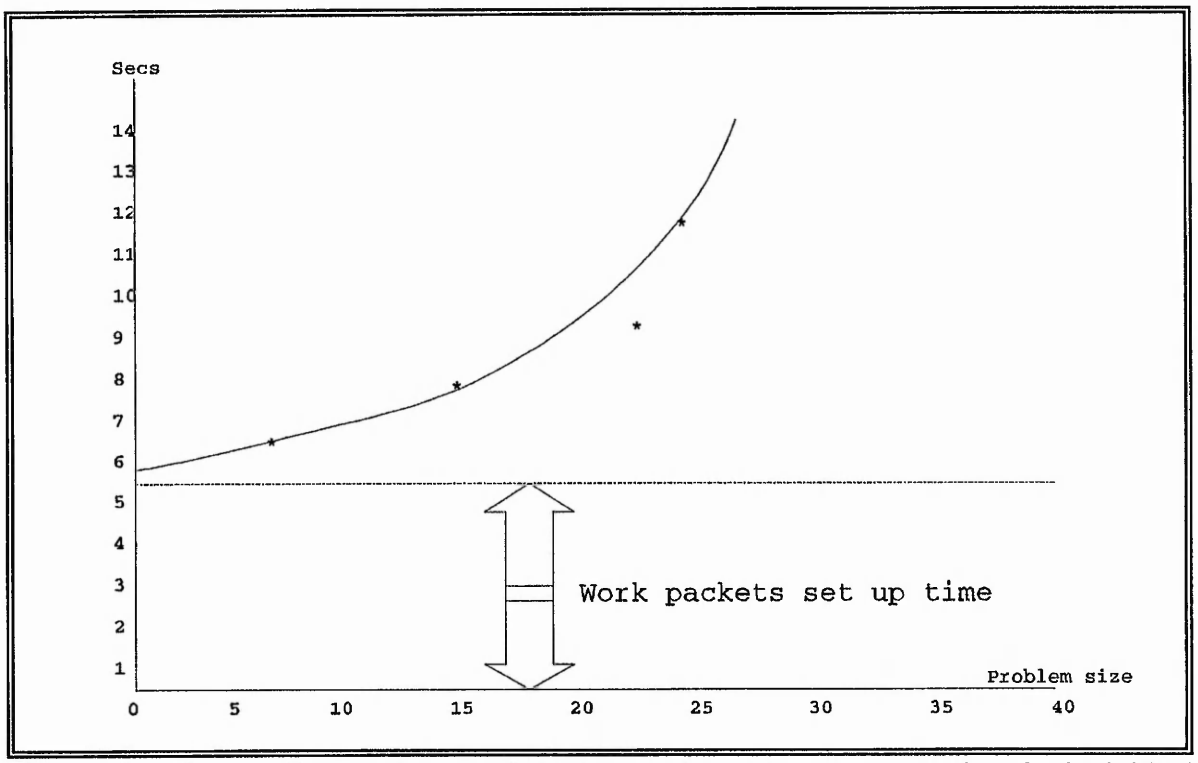

Figure 58: The coordination time graph (including the work packet setup time for individual subnetworks) of a 65 node network.

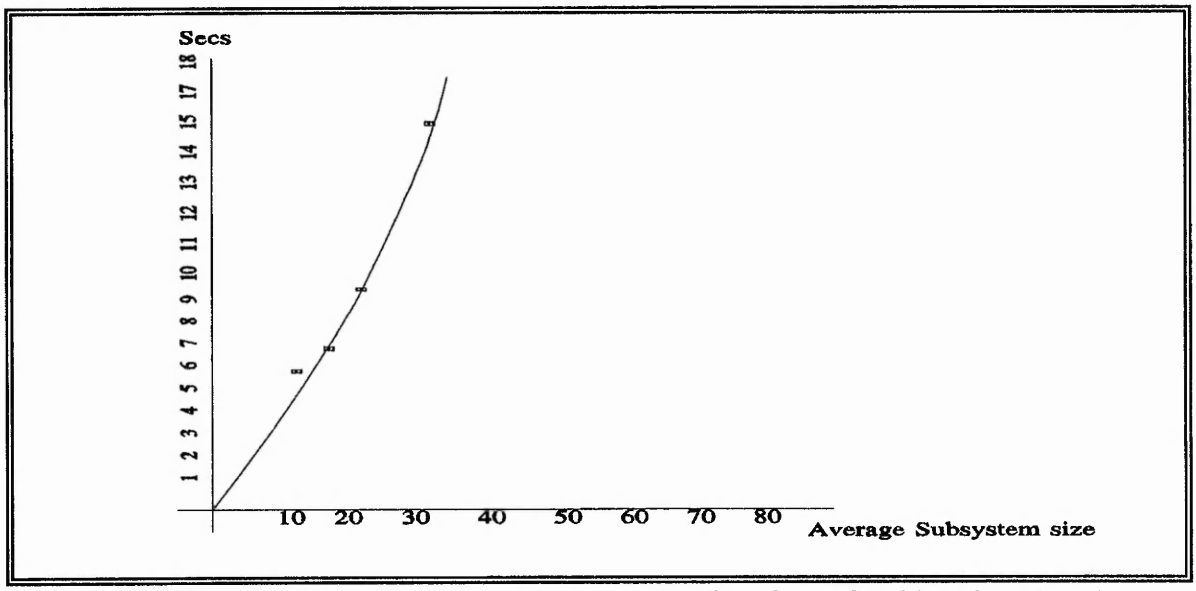

Figure 59: This graph represents the subsystem solution time of a 65 node network.

Table 7(a): *65 node network partitioned into 5 subnetworks(4 iteration).*

Subnetwork Number	Subsystem solution times(Secs)	Average number of nodes / subnetwork	Number of cut-branches	Coordination time(Secs)
1	8.123	13	16	13
2	7.2			
3	7.15			
4	6.0			
5	5.12			

Table 7(b): *65 node network partitioned into 4 subnetworks(4 iteration).*

Subnetwork Number	Subsystem solution times(Secs)	Average number of nodes / subnetwork	Number of cut-branches	Coordination time(Secs)
1	10.12	16	14	9.7
2	9.32			
3	6.78			
4	5.3			

Table 7(c): *65 node network partitioned into 3 subnetworks(4 iteration).*

Subnetworks Number	Subsystem solution times(Secs)	Average number of nodes / subnetwork	Number of cut-branches	Coordination time(Secs)
1	12.12	21	11	8.025
2	13.3			
3	7.0			

Table 7(d): *65 node network partitioned into 2 subnetworks(4 iteration).*

Subnetworks Number	Subsystem solution times(Secs)	Average number of nodes / subnetwork	Number of cut-branches	Coordination time(Secs)
1	15.025	32	5	5.67
2	16.25			

Table 8: *Table of results for a 65 nodes network (partitioned into 2 - 5 subsystems) implemented in a distributed Virtual Node environment.*

Number of Subnetworks	Number of cut-branches	Network size	Subsystem solution time(Secs)	Coordination time(Secs)
2	5	65	15.5	5.67
3	11		10.7	8.025
4	14		7.75	9.7
5	16		6.6	13.0

6.4 Conclusion

This chapter was concerned with design and development of a distributed computing environment, for distributing the water system simulation program, using the Ada language. The partitioned Jacobian approach was superseded by the more efficient approach whereby the network partitioning and linearization are performed on the individual processors thus reducing the amount of data transfer between the subsystem workers and the coordination routine.

The concept of "virtual node" provided an abstraction of the target physical processing node in Ada, therefore enabling distributable components of a system to be coded and then mapped onto the underlying distributed computing system.

As mentioned above, the distributable components comprised of coordination and subsystem solution processes. The process of coordination is static therefore it was best represented as a unique virtual node. On the other hand, the subsystem solution process comprised of a collection of independent programs varying in numbers according to the partitioning scheme employed, and was programmed as Virtual Node types. Consequently if a greater number of subnetworks is appropriate for a given network new worker tasks can be instantiated from the already defined Virtual node base type.

Moreover, water distribution system is periodically subject to expansion therefore requiring more processing power, the extra processing power needed is easily provided by simply instantiating new tasks of worker virtual node type on a separate processing node in the target distributed computing system.

The inter-virtual node communication is provided by remote entry call, an extended version of Ada's rendezvous mechanism incorporating the remote call mechanism. An added advantage of using familiar mechanism for inter-virtual node communication is the fact that the remote call is processed on the machine of the callee, thus providing a clearer picture of the work-load of each of the physical nodes, enabling a more even distribution. The remote rendezvous mechanism forms a layer of software which is

separately developed from that of the application program layer.

The problem of virtual node types sharing objects defined in the non-template packages can be avoided by ensuring that every instance of the virtual node type is assigned to a single physical node. Even if the virtual node type had to share the physical processing node, it should be grouped with unique virtual nodes. However, the remaining problem was that of access to remotely visible entries. To overcome this problem the interface task of the virtual node type was placed within a template package, then an access type was defined as a reference to the interface task type. The access values of the interface task of the subsystem virtual node type can now be used by the coordination virtual node to call the appropriate instance of the subsystem worker tasks.

This research has established a platform for the development of a distributed computing system based on the generalisation of ADA-rendezvous mechanism. A standardising layer cooperating with the remote rendezvous layer was created to insulate the application program development stage from the differences that may exist between different computing environment's communication systems. Thus the explicit use of communication primitive at remote rendezvous layer was not required. To create communication end-points(sockets) a generic package defined within the standardising layer is instantiated whose body consequently create the communication end-points dynamically and deallocate them after use.

To solve the problem of type conversion during the transmission of data packets over the communication network, discriminant record types were introduced. Therefore, no explicit data conversion was necessary since the conversion would take place automatically, thus alleviating the need for the use of UNCHECKED_CONVERSION package.

The distributed water system simulation program was tested on realistic networks. The distributed computing system environment consisted of a federation of SUN/SPARC workstation having large local memory with UNIX operating system. The results obtained

seems to indicate that the coordination time is quasi-quadraticly dependent on the problem size. on the other hand the pseudo-linearity of the subsystem solution times indicates that an increase in the subsystem size would almost inevitable result in an increase in the subsystem solution time.

This research has demonstrated that the concept of "virtual node" for modelling the distributed processing nodes of a loosely-coupled system, is a viable approach(Hosseinzaman and Bargiela**[61]**). The DIADEM**[5]** approach was selected amongst other methods**[68,71,25,4]** to represent the distributed water system simulation program because: (i) it offers a compiler independent virtual node approach. Therefore conventional Ada compilers can be used to develop the distributed application program, (ii) it allows the notion of virtual node type. which is particularly useful in this implementation since it offers dynamic creation of processing nodes to fit a particular partitioning scheme, and (iii) it uses a rendezvous like mechanism for inter-virtual node communication - useful since rendezvous mechanism gives a better reflection of the work-load of the processing nodes in the system.

CHAPTER 7: Conclusion and Further Research.

7.1 Conclusion.

The early water distribution systems employed basic measurement devices to monitor and control the system. Monitoring of system parameters was confined to reservoir's water level or pump status, the distribution network itself was largely unmonitored because of the difficulty of interpreting the measurement data by the human operator due to the enormity of the collated system data. Clearly automation were needed to assist the operator in monitoring the water network. On the other hand, the continued price decreases in both required hardware and software meant that the automation of the system monitoring, in order to achieve optimal control, could successfully and economically be performed.

The initial successes in the simulation of water distribution systems prompted the industry to consider the development of on-line decision support systems which would integrate existing telemetry systems and simulation software. Attempts made to accomplish the undertaking of developing on-line decision support system ended in failure, because of the rapid increase of the computational requirements of the simulation algorithms with increase of the physical network size.

To overcome such problems, sparsity exploiting techniques were employed in the simulation. The sparsity-directed techniques exploit the inherent sparsity of water distribution systems in order to achieve speedup. These techniques were discussed in Chapter 2. They include: sparsity-directed matrix inversion routines - Bi-factorization and Bartels_Golub decomposition, sparsity-directed storage techniques - "Link-Lists", and near-optimal pivoting techniques.

However, the results of our investigations indicate that, the efficiency of the techniques employed in solving the derived system matrix is highly dependent on two factors, firstly is the pivotal strategy used in order to achieve minimum "fill-in" thus reducing storage requirements and achieving speedup, and secondly the storage technique itself. However, despite the fact that, using these techniques may reduce the computational requirements of

the simulation software in the short term, but in the long term they cannot meet the growing computational requirements due to periodic expansion of the network. Hence alternative solution methods have to be found.

Chapter 3 addresses this problem and focuses on the parallel processing algorithms. The research resulted in the development and implementation of a nonlinear diakoptics technique. The objective was to partition the system into a number of parts, so that each part is isolated from the rest of the system. The component networks are solved independently on different processing units simultaneously, and the solution of the complete network is obtained by coordinating the component network's solutions. Two methods were employed for obtaining the component network's solutions - "the partitioned system matrix" after linearization and "the partitioned system matrix" before linearization. However, while the linearized partitioned system matrix method 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 amount of data associated with each jacobian block. Thus partitioning before linearization approach was chosen instead.

The algorithm was initially run on a closely-coupled(PC-based Transputer system configured in "Tree" and "Pipe-line" modes) computing system. The results indicate that the processor topology has little effect on the program's execution time. Contrary to Bowden **[16]**, storage problems were not encountered. The computational efficiency of the algorithm was evaluated using two realistic networks and results were extrapolated to large scale systems(i.e. 1300 nodes). The results obtained indicate that the time required to coordinate subsystem solutions depends almost quadraticly on the coordination problem size, on the other hand, the subsystem solution times were found to depend quasi-linearly on the number of nodes in the subsystem.

The overall computational efficiency and storage requirement of the network tearing algorithm was found to be strongly influenced by the way the system is partitioned. In the cases where the system has a simple layout, a fairly good cluster partition can be

achieved by inspection, while for complex systems an algorithm must be used to systematically partition the graph into an optimal arrangement of clusters.

This research resulted in the development of a new nonlinear parallel processing algorithm (nonlinear diakoptics).

Chapter 4 describes two graph partitioning techniques with different theoretical basis. The first method known as the "greedy algorithm", has its basis in a heuristic derivation. The second method known as "simulated annealing", however, is theoretically based on the analysis of combinatorial optimization problems. The application of the heuristic method was found to be strongly problem bound, thus cannot be generalized for use in broader range of partitioning problems. On the other hand, the simulated annealing algorithm is generally applicable to a wide range of graph partitioning problems. However, an important conclusion from the results of the computer simulation, indicates that, the performance of the algorithm strongly dependent on the selected cooling schedule. As far as the performance of the algorithm is concerned, it can be concluded that for our implementation the quality of the solution obtained by the simulated annealing algorithm is at least as good as (and sometimes better) than the results obtained from the application of heuristic algorithms(greedy).

Chapter 5 covered issues concerning the water distribution system topology and demonstrated how they can be resolved using a distributed computing model. One of the primary requirements of a water distribution system is the ability to expand. Expansion of the network is closely mirrored by the increased complexity of its mathematical model thus requiring more computational power. The distributed computing model presented consisted of two types - namely the tightly-coupled and the loosely-coupled systems. The most suitable type satisfying the aforementioned requirements of the water distribution system was to be that of the loosely-coupled system.

Moreover, in order to develop application programs for loosely-coupled systems, the programming language had to have certain characteristics. These are: software

configurability, inter-process communication mechanism(synchronous or asynchronous) and finally the partial failure mechanism. The suitability of each language would be measured on how many of these requirements it can satisfy. The Ada language satisfies most of these requirements, however, it is generally acknowledged that the language support in the area of distributed systems is lacking.

This research has established a framework for the development of a distributed computing system based on the generalisation of the ADA-rendezvous.

Several strategies were devised**[23,4,138,30,24,25,71,132,84,36,131,68,5]** to overcome the problems associated with lack of language support for the development of large truly distributed software. These strategies tend to agree that best approach to program distributed applications in Ada, is pre-partitioning of the application program. On the subject of inter-process communication mechanism two methods predominate - namely the Remote Entry Call and Remote Procedure Call. There are strong arguments for and against using both of these methods. However, the conclusion here is that, the application program inter-process communication needs, should be the deciding factor.

Chapter 6 has documented the implementation of a distributed computing environment, for water system simulation,which served to validate the general concept of distributed computation using ADA Virtual Nodes. Major functional units of the system were identified and diagrammatically defined - namely the coordination and the subsystem solution routines. The partitioned Jacobian approach was superseded by the more efficient approach whereby the network partitioning and linearization are performed on the individual processors thus reducing the amount of data transfer between the subsystem workers and the coordination routine.

The concept of "virtual node" for modelling the distributed processing nodes of a loosely-coupled system, has been implemented and demonstrated to be a viable approach . The DIADEM**[5]** approach was selected amongst other methods**[68,71,25,4]** to represent the distributed water system simulation program because: (i) it offers a compiler

independent virtual node approach. Therefore conventional Ada compilers can be used to develop the distributed application program, (ii) it allows the notion of virtual node type. which is particularly useful in this implementation since it offers dynamic creation of processing nodes to fit a particular partitioning scheme, and (iii) it uses a rendezvous like mechanism for inter-virtual node communication - useful since rendezvous mechanism gives a better reflection of the work-load of the processing nodes in the system.

7.2 Suggestions for Further Research.

A new nonlinear network tearing algorithm is introduced in this thesis. The algorithm's applicability to a wider range of nonlinear engineering problems should be explored and compared with its application to problems associated with the simulation of large water distribution systems. Examples of nonlinear systems which could benefit from the newly developed nonlinear network tearing algorithm developed in this work includes; electricity distribution systems, gas distribution networks, traffic management systems, and industrial process control systems.

The future trend in the area of water distribution and control systems, depends on the availability of methodologies for integrating the existing technologies such as Distributed Control and SCADA systems to develop a more sophisticated on-line decision support systems. The introduction of X-windows is a step in the right direction for providing such environments. Another key to easier integration, beyond the hardware, the operating systems, the communication and databases, is Object-Oriented programming**[58]**. Thus a comparison can be drawn between the object-oriented methodology and the Virtual Node approach introduced in this thesis. Although one may argue that they are not exactly the same, but the virtual node concept could be interpreted as a starting point in solving the problem of integrating the available technologies.

The introduction of Ada9x, which is a fully Object-Oriented language, could be viewed as the ultimate tool for developing these objects. Furthermore, the Ada9x "partition" mechanism adopts many of the characteristics of the Virtual Nodes, therefore allowing for an easier transition between software languages, if it is decided to use Ada9x as the sole programming environment for development and integration of such technologies.

REFERENCES

[1] "Ada Language Reference Manual", Jan. 1983, ANSI/MIL-STD 1815A.

[2] -, "Alsys Ada compiler RM", Alsys ltd,1992.

[3] Amari,S.,"Toplogical Foundations of Kron's Tearing of Electrical Networks," R.A.A.G. Memoirs, 3, pp.322-349,1962.

[4] Armitage,J.W, and Chelini, J.V., "Ada Software on Distributed Targets: A Survey of Approaches", Ada Letters, 1985, 4(4), pp. 32-37.

[5] Atkinson,C.,Moreton,.T,and Natali,.A, "Ada for Distributed Systems", 1988, Cambridge Univ. Press.

[6] Bal. H, "Programming Distributed Systems",1990, Prentice Hall.

[7] Bargiela,A.,"On-line Monitoring of Water Distribution Networks", PhD Thesis, University of Durham,1984.

[8] Bargiela,A., AlDabass,D.,"A simulated real-time environment for verification of advanced water network control algorithms", 5th System Science Conference, Wroclaw, Sept.1986.

[9] Bargiela,A.,"Opimal telemetry system for water networks", Int. Symposium on optimal Modeling of Water Distribution Systems, Lexington, K.Y., May 1988.

[10] Bargiela,A., Hainsworth,G., "A graphical interactive software tool for confidence limit analysis",AWWA Conference on Computer and Automation in the Water Industry, Denver, CO, 2-4 April 1989.

[11] Bargiela,A.,Hainsworth,G.,"Pressure and flow uncertainty in water systems," ASCE Jou. of Water Resources Planning and Management, Vol. 115, pp.212-229,1989.

[12] Bargiela,A.,"Nonlinear Network Tearing Algorithm for Transputer System Implementation" Proc. TAPA' 92, pp.19-24, Nov. 1992.

[13] Berry,R.D, "An optimal ordering of electronic circuit equations for a sparse matrix solution," IEEE Tans. Circuit Theory, Vol. CT-18, pp. 139-145, 1971.

[14] Bertele,U., and Brioschi,F.,"On the theory of the elimination process," J. Math. Anal.Appl., Vol. 35,pp. 48-57,1971.

[15] Birrel, A.D, and Nelson,R.J, 1984, "Implementing Remote Procedure Calls",ACM Trans. Comput. Syst.,Vol.2, Feb 1984,Assoc. for Computing Machinary Inc.

[16] K.Bowden,"Kron's Method of Tearing on a Transputer Array", The Computer Journal, Vol. 33,No.5,1990.

[17] Bozzi,U.,and Tiramani,A.R,"Techniques for reordering equations for the solution of sparse linear systems,"Calcolo, Vol.9, pp. 139-142, 1972.

[18] Brameller,A., John,M.N,Scott,M.R, "Practical Diakoptics for Electrical Networks," Chapman and Hall Ltd, 1979.

[19] Burns,A. et. al. "A Review of Ada Tasking", YCS.78, Dept. of Computer Science, University of York, 1985.

[20] Canal,M.,"Elimination ordonees,un processus dimuant le volume des calculs dans la resolution des systemes lineaires a matrice creuse," in Proc. Power Systems Computation Conf. London, 1963.

[21] A.Cholerton."Ada for Closely-coupled Multi-processor Targets", In Proceedings, TRI-Ada'89,PA,Oct.23-26,1989.

[22] R.P.Cook:"* MOD - A language for Distributed programming," Proceedings of the 1st International Conference on Distributed computing systems, Huntsville,Alabama,pp.233-241(Oct.1979).

[23] Cornhill,D.,"Four approaches to partitioning Ada programs for execution on Distributed Targets", 1984,IEEE Computer Soc. 1984 Conf. on Ada Applications and Environments, pp. 153-164.

[24] Cornhill,D., "A survivable Distributed Computing System for Embedded Application Programs Written in Ada", Ada Letters, 1983(Nov/Dec).

[25] Cornhill,D., Beaue,J. and Silverman,J., "Preliminary Reference Manual for the Ada Program Partitioning Language", Honeywell(Jan 1983).

[26] Coulbeck,B.,Sterling,M.,"Optimised Control of water distribution systems," IEE Proc. Vol.125, Oct.1978.

[27] Crichlow,J.M,"An Introduction to Distributed and Parallel Computing", 1988, Prentice Hall International(UK) Ltd.

[28] Dantzig,G.B, Harvey,R.P, Mcknight,R.D, and Smith,S.T,"Sparse matrix techniques in two mathematical programming codes," in [585, pp.85-99].

[29] Dantzig,G.B, Wolfe,P.,"Decomposition Principle for linear Programs," Operation Research, 8(1), pp. 101-111,1960.

[30] Darpra, Maderna, stammers, et. al. ,"Using Ada and APSE to support Distributed

Multi_microprocessor Targets", Ada letters, 1984, Vol.3, No.6.

[31] Davies, D.W. et al, 1979,"Computer Networks and their protocols", Chichester: John Wiley & Sons.

[32] Davies,D.W. and Barber,D.L.A, 1973,"Communication Networks for computers", London: John Wiley & Sons.

[33] Dahlguist,G.,and Bjork,A., "Numerical methods," Englewood cliffs, NJ; Prantice-Hall, 1974.

[34] Dembart, B., and Erisman,A.M,"Hybrid sparse matrix methods," IEEE Trans. Circuit Theory, Vol. CT-20, pp. 641-649, 1973.

[35] Dillingham, J.H, "Computer Analysis of Water Distribution system part II," water and sewage works, Feb.1967, pp. 43-45.

[36] Downes,V.A, and Goldsack,S.J.,"The use of the Ada Language for programming a distributed system",in V.H. Hoase, Proc. IFAC/IFIP workshop on Real-time programming, pp. 39-44, pergamon Press, Oxford 1980.

[37] Duff,I.S, and Reid,J.K, "Some design features of a sparse matrix code," ACM Trans. Math. Software, Vol.5, pp. 18-35, 1979.

[38] Duff,I.S, and Reid,J.K, "A comparison of sparsity orderings for obtaining a pivotal sequence in Gaussian elimination," J. Inst. Math. Appl., Vol 14, pp. 281-291,1974.

[39] - , "Pivoting for size and sparsity in linear programming inversion routines," J. Inst. Math. Appl., Vol. 10, pp. 289-295, 1972.

[40] - , " On the product form of inverses of sparse matrices and graph theory," SIAM Rev., Vol. 9,pp. 91-99,1967.

[41] - , "Solution of a system of simultaneous linear equations with a sparse coefficient matrix by elimination methods," BIT, Vol.7,pp.226-239, 1967.

[42] Duff,I.S, "A survey of sparse matrix research," Proc. IEEE, Vol.65, No4, Apr. 1977.

[43] Duff,I.S, and Reid,J.K, J. Inst. Maths. Applics. 1976.

[44] Edelmann,H., "Optimal strategies in the direct solution of the systems of linear equations with sparse matrices," Z.Agnew. Math. Mech.,Vol. 45, pp. 13-18,1965.

[45] Edmonds,J.,"Matroids and the Greedy algorithm," Maths. Programming, Vol.1,pp.127-136, Nov.1971.

[46] Forrest,J.J.H, and Tomlin,J.A,"Updating triangular factors of the basis to maintain sparsity in the product form simplex method," Mathematical Programming, 2(1972), pp. 263-278.

[47] Garey,M.R, and Johnson,D.S,"Computer and Intractability," Freeman, San Francisco, 1979.

[48] Gentelman,W.M, 1973, J. Inst. Maths Applics, Vol.12, pp. 329-336.

[49] Geoffrion,A.M, "Solving Bicriterion Mathematical Programs Operation Research,Vol. 15, pp.39-54, 1967.

[50] Gill,P.E, and Murray,W., 1973 Linear Algebra Applics., Vol.7,pp.99-138.

[51] Gilland,P.E, and Murry,W.,"A numerically stable form of the simplex algorithm," Linear Algorithm Appl., 7(1973), pp. 99-138.

[52] Givens, J.Inst. Maths. Applics., 1976, Vol.17, pp. 267-280.

[53] Golub,G. 1965. Num. Math. Vol.7, pp. 206-216.

[54] Golub,G., 1969, Information Processing, Vol.68,North Holland.

[55] Gustavson,F.G,"Some basic techniques for solving sparse systems of linear equations," in Proc. Conf. at IBM Research Center,NY,Sept. 1971,pp.41-52.

[56] - ,"FORTRAN subroutines for handling sparse linear programming bases," HMSO, London, England, AERE Rep. R.8269,1976.

[57] Happ,H.H, "Diakoptics and Piecewise methods," IEEE Trans. PAS-89, 1970, pp.1373-1382.

[58] M.Heitz,B.Labreville,"Design and Development of Distributed Software Using HOOD and Ada", Proc. of Ada Europe Int. Conf., 1988,pp143-156.

[59] Himmelblau, D.M.:"Decomposition of Large_Scale systems," Chem. Eng. Sci. 21,pp.425-438, 1966,Chem. Eng. Sci. 22, pp.883-895, 1967.

[60] Hosseinzaman,A. and Bargiela,A.,"Parallel simulation of Nonlinear Networks, using Diakoptics," Proc. of Int. Conf. on Parallel Computing and Transputer Appl. 1992, Barcelona, Spain, pp.1463-1473.

[61] Hosseinzaman,A. and Bargiela,A.,"ADA's Virtual Node based Water System Simulator", Proc. of ADA UK International Conference, ADA UK JOU., April 1994,

London,England.

[62] Householder,A.S, "The Theory of Matrices in Numerical Analysis," Blaisdell, 1964, New York.

[63] Hsieh,H.Y, and Ghausi,M.S, "On optimal pivoting Algorithms in sparse matrices," IEEE Trans. Circuit Theory(corresp.), Vol.CT-19,pp.93-96,1972.

[64] Hsieh,H.Y, and Ghausi,M.S, "On sparse matrices and optimal pivoting algorithms," IBM Tech. Rep. TR 22.1249.

[65] Hsieh,H.Y,"Pivoting-order computation method for large random sparse systems," IEEE Trans.Circuits and Systems, Vol. CAS-21,pp. 225-230,1974.

[66] - , "A probabilistic approach to optimal pivoting and prediction of fill-in for random sparse matrices," IEEE Trans. Circuit Theory, Vol. CT-19, pp. 329-336,1972.

[67] Hu,T.C,"Integer Programming and Network Flow," Reading MA, Addison_Wesley, 1969.

[68] Hutcheon,A.D, and Wellings,A.J.,"Ada for Distributed Systems",1987, Computer Standards and Interfaces,(North- Holland).

[69] Irving,M.R.,and Sterling,M.J.H,"Optimal Network Tearing using Simulated Annealing", IEE Proc., Vol.137,Pt.C,No.1,Jan 1990.

[70] Isaac,L.T and Mills, K.G (1980), "Linear theory method for Pipe network analysis," Jou. Hydr. Div.,ASCE,106(7),1191-1201.

[71] Jessop,W.H,"Ada Packages and Distributed Systems", SIGPLAN Notices 17(2),pp. 28-36(Feb 1982).

[72] Jowitt,P.,Xu,C.,"Optimal valve control in water distribution networks," ASCE Jou. of Water Resources Planning and Management, Vol. 116, pp.455-472,1990.

[73] M.Kamrad,R.Jha and G.Eisenhauer,"Reducing the complexity of reconfigurable systems; Ada", 2nd International Workshop on Real Time Ada issues, 1988.

[74] M.Kamrad,R.Jha and D.cornhill,"Distributed Ada",ACM,1987.

[75] Key,J.E,"Computer program for the solution of large sparse unsymmetric systems of linear equations," Int. J. Numer. Methods Eng., Vol. 6, pp. 497-509, 1973.

[76] Kirkpatrick,S., Gellatt,C.D, and Vecchi,M.P,"Optimization by simulated annealing," Science, 1983,220, pp.671-680.

[77] J.Knight and M.Rouleau,"A new approach to fault tolerance in distributed Ada programs,"2nd International Workshop on Real Time Ada issues, 1988.

[78] Knight,J.C, and Urquhart,J.I.A, "On the implementation and Use of Ada on Fault_Tolerant Distributed Systems", 1984, ADA Letters, pp. 53-64.

[79] Knuth,D.E, The Art of Computer Programming. Vol. Fundamental Algorithms. Reading, MA: Addison-Wesley, 1965.

[80] Kron,G.: Diakoptics:"The Piecewise solution of Large_Scale systems," Mcdonald, London, 1963.

[81] Kulikowski,R.,"Optimization of Large_Scale systems," Fourth congress of IFAC, Warszawa, 1969, survey paper 16,pp. 1-40.

[82] B.W.Lampson: "Remote Procedure Calls," pp.365-370, Vol.105,1981.

[83] Lasdon,L.S,"Optimization Theory for large systems," Macmillan Co. London,1970.

[84] Liskov, B., "Linguistic Support for distributed programs", IEEE Transactions on Software Eng., SE-8,No.3,May 1982.

[85] Lorin,H., "Parallelism in Hardware and Software: Real and apparent concurrency", 1972, Prentice Hall Inc., Englewood Cliffs, New Jersey.

[86] Markowitz,H.M, "The elimination form of the inverse and its application to linear programming," Management Sci. , Vol.3, pp. 255-269, 1957.

[87] May,D. and Shepherd,R.,"The Transputer Implementation of occam",1984,Proc. Int. Conf. on Fifth Generation Computer Systems, pp. 533-41, Tokyo.

[88] Mesarovic,M.D,Macko,D.,Takahara,Y.,"Theory of Hierarchical Multilevel systems," Acad. Press, New York, London, 1970.

[89] Metropolis,N., and Rosenbluth,A.W,"Equation of state calculations by fast computing machines," J.Chem. Phys.,1953, Vol.21,(6),pp.1087-1092.

[90] Mine,H.,Ohno,K.,"Decomposition of Mathematical Programming Problems by Dynamic Programming," J. Math. Anal. and Appl., Vol.32, pp.370-385,1970.

[91] Morris,D.S, Wheeler,.T, "Distributed Program Design in Ada: An example", IEEE Comput. Soc. 2ND INT. CONF. on ADA Applications and Environment, 1986, pp. 21-29.

[92] Munkres,J.,"Algorithms for assignment and transportation problems," J.SIAM,Vol.

5, pp. 32-38, 1957.

[93] B.J.Nelson: "Remote Procedure Call", CMU-CS-81-119, Detartment of Computing Science, Carngie-Mellon University(May1981).

[94] Nemhauser,G.L,"Decomposition of linear programs by Dynamic Programming," Naval Res. Logist. Quart. 11, pp.191-195, 1964.

[95] Onederra,R.,"Diakoptics and codiakoptics of electrical networks, R.A.A.G, Memoirs 2, pp.369-388,1958.

[96] Orchard-Hays,W., Advance linear programming computing Techniques. New York: McGraw-Hill,1968.

[97] - ,"Analysis of sparse systems," D.Phil. thesis, Oxford Univ., Oxford,England,1972.

[98] Page,E.S,and Wilson,L.B, Information representation and Manipulation in a computer. Cambridge, England: Cambridge Univ. Press, 1973.

[99] - , "The solution of large sparse unsymmetric systems of linear equations," J. Inst.

Math. Appl., Vol. 8, pp. 344-353, 1971.

[100] - ,"The solution of large sparse unsymmetric systems of linear equations,"J. Inst. Math. Appl.,203 pp.1240-1245,1971.

[101] - ,"The implementation and use of sparse matrix techniques in general simulation programs," Comput. J., Vol.17,pp.165-170,1974.

[102] - ,"Symbolic generation of an optimal Crout algorithm for sparse systems of linear equations," J. Assoc. Comput. Mach., Vol. 17,pp. 87-109,1970.

[103] Peters,G., and Wilkinson,J.H, 1970, Comput. J., Vol. 13, pp. 309-316.

[104] - "Extended application of the sparse tableau approach-Finite elements and least squares," Elec.Sci. and Eng. Dep. and Comput. Sci: Dep., UCLA, Tech. Rep.,1974.

[105] - ,"Iterative refinement of linear least squares solutions I," BIT, Vol.7, pp. 257-278,1967.

[106] Peters,G.,and Wilkinson,J.H, 1969, Information Processing, Vol.68,North Holland.

[107] Phillips,W.,"The storage and the inversion of large sparse matrices," Math. and

Stats. Group,ICI Wilmslow Rep.,1970.

[108] Powell,M.J.D, and Reid,J.K, 1969, In Inform. Processing, Vol.68, North Holland,pp.122-126.

[109] A.J.Purves and A.L.Cesario, "Computer Application in Water Industry", Water/Engineering & Management, 1993.

[110] Ralston,A., and Wilf,H.S, "Mathematical methods for Digital computers," J.Wiley and Sons, Inc. New York, 1960.

[111] Ranks,S.,Won,Y. and Sahni,S., "Programming a Hypercube Multicomputer", 1988,IEEE Software, Vol. 5, No.5, pp. 69-77.

[112] Reid,J.k,"A sparsity_Exploiting variant of the Bartels_Golub Decomposition for linear programming Bases," Aug.1975, computer Science and system division, Harwell.

[113] Rose,D.J, and Tarjan,R.E, "Algorithmic aspects of vertex elimination," in Proc. 7th Ann. ACM Symp. on the Theory of computing, pp. 245-254, 1975.

[114] Roth,J.P,"An application of Algebraic Topology to Numerical Analysis, on the existence of a solution to the network problem," Proc. Nat. Academy of Sciences, Vol.41,pp.518-521.

[115] Sakamoto,A.,Shirakawa,I., and Ozaki,H.,"Ordering of pivot operations on sparse systems of equations," Information processing in Japan, Vol. 12,pp. 84-89,1972.

[116] Samch,A.H, "On some parallel algorithm on a ring of processors," comput. Phys. comm. Vol. 37, pp 159-166,1985.

[117] Sangiovanni_Vincebtelli,A. and Chen,L.,"An Efficient Heuristic Cluster Algorithm for Tearing Large_Scale Networks," IEEE Trans. CAS, Vol. CAS24,1977.

[118] Sato,N., and Tinney,W.F, "Techniques for exploiting the sparsity of the network admittance matrix," IEEE Trans. Power, Vol. PAS-82,pp. 944-949,1963.

[119] Sato,N. and Tinney,W.F, "Techniques for exploiting the sparsity of the network admittance matrix," IEEE Trans. on Power Apparatus, and sys. 82(1963), pp 944-950.

[120] Saunders,M.A, "Large_Scale linear programming using the Cholesky factorization," Report STAN_CS_72_252(Stamford Univ., 1972).

[121] A.M.Schulte & A.P.Malm, "Integrating Hydraulic Modeling and SCADA systems for system planning and control",Water/Engineering & Management,1993.

[122] Sechen,C., and Sangiovanni_Vincentelli,"The timber_Wolf placement and routing

package," in Proc. 1984 custom Integrated Circuit Conf.(Rochester NY), May 1984.

[123] Siegel,I.H,"Deferment of computation in the method of least squares," Math. Comput., Vol.19, pp.329-331, 1965.

[124] - ,"A comparison of some methods for the solution of sparse overdetermined systems of linear equations," J. Inst. Math. Appl., Vol. 17, pp. 267-280, 1975.

[125] Seitz,C.L., "The Cosmic Cube", 1985, communication ACM, Vol.28, No.1,pp.22-33.

[126] Sheild,J.,"Partitioning Concurrent VLSI simulated programs onto a multiprocessor by simulated annealing," IEE Proc. Vol.134, Pt.E,No.1 ,Jan.1987.

[127] M.Sloman,J.Magee and J.Kramer:"Building Flexible Distributed Systems in Conic",pp.86-105 in Dist. Computing Systems program, ed. D.A. Duce,Peter Peregrinus Ltd.(1984).

[128] Sorenson,D.C,"Analysis of Pairwise pivoting in Gaussian elimination," IEEE Trans. comput., Vol. C-34, pp 274-278, 1985.

[129] 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.

[130] Stevens,W.R.,"Unix Network Programming", 1991, Prentice-Hall,Inc.

[131] Tanenbaum,A.S,"Computer Networks", 1981, Prentice/Hall International.

[132] Tedd,M., Crespi-Reghizzi,S., and Natali,A., " Ada for Multi_microprocessors, The Ada companion Series, Cambridge University Press(1984).

[133] Tinney,W.F, and Walker,J.W, "Solution of sparse network equations by optimality ordered triangular factorization," Proc. IEEE, 55 1967, pp 1801-1809.

[134] Tinney,W.F, "Some examples of sparse matrix methods for power network problems. In Proc. of the 3rd Power systems comput. conference. Rome 1969.

[135] Tinney,W.F, "Optimal ordering for sparsely coupled subnetworks," Bonneville Power Administration, Portland,OR, Internal Rep., 1968.

[136] Tosovic,L.B, "Some experiments on sparse sets of linear equations," SIAM J. Appl. Math., Vol.25,pp.142-148,1973.

[137] Vinchenevetsky,R.,"Serial solution of Parabolic Partial Differential Equations," Simulation, Vol.13, pp.47-48, 1969.

[138] Volz,R.A., et. al., "Some Problems in Distributing Real_time Ada Programs across

machines",1985, Ada in use, Proc. of the Ada Int. Conf., Paris , pp. 72-84.

[139] R.A.Volz,P.kirshnan and R.Theriault,"Distributed Ada - A case study", In Distributed Ada 1989, Proceeding of Symposium,pp.17-59,1989.

[140] Walski, T.M, - Analysis of water distribution system,VNR - New York.

[141] A.J.Wellings et.al.,"Communication between Ada programs," IEEE Trans.,pp145-152,1984.

[142] Wilkinson,H., and Reinsch,C., Handbook of Automatic computation . Vol.2, New York:Springer Verlag,1971.

[143] Wilkinson,J.H, "Error analysis of direct methods of matrix inversion," J.ACM, Volume 8, pp. 281-330,1961.

[144] B.Williams,"Integrating SCADA systems - Applications and Technology",Control Systems, Nov.1994.

[145] Wood,D.J, "Computer Analysis of Water Distribution system", Journal of the HYDRAULICS DIVISION _ Proceeding of the American Society of Civil Eng. July 1972.

[146] Zolenkopf,K., "large sparse sets of linear equations," Academic Press, 1971.

[147] -,4th International Workshop on Real Time Ada issues, Ada letters, Vol. x, No.9,1990.

[148] "ADA-to-C" interface release notes, Alsys Ltd,199_ .

[149] Parallel FORTRAN, User Guide, Version 2.1.4, 3L Ltd.