Multi-microprocessor power system simulation

J. W. Flaxman

MULTI-MICROPROCESSOR POWER SYSTEM SIMULATION

A Ph.D. THESIS

by

J.W. FLAXMAN

UNIVERSITY OF DURHAM

SCHOOL OF ENGINEERING AND APPLIED SCIENCE

1987

MULTI-MICROPROCESSOR POWER SYSTEM SIMULATION

A Ph.D. THESIS by J.W. FLAXMAN 1987

ABSTRACT

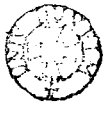
This thesis presents the results of research performed into the simulation of electrical power systems using a set of microprocessors operating in parallel . The uses and methods of simulation on analog and single processor computers are discussed as well as on multiple processor machines . It then considers various methods already used in the field of simulation for both the dynamic and network sets of equations in detail and the problems of using them on parallel processors . Several possible methods of parallel simulation are proposed and the best of these developed into a detailed algorithm for simulating both the dynamic and network portions of the power system .

The different types of multiprocessor system are looked at , both in terms of physical configuration and the type of hardware used to implement the different types of system . The problems inherent in parallel computing are discussed and a form of multiprocessor, suitable for the simulation algorithm, is then developed taking these problems into account . The hardware is developed using widely available hardware and the algorithm is then implemented upon this hardware .

The results obtained using the simulator show that the proposed system provides a more economical solution, both in terms of the time taken in producing results and in the cost of the system, when compared with a conventional single processor computing system such as a mini computer .

ACKNOWLEDGEMENTS

I would like to express my thanks to those people, without whom my research and this thesis would have been impossible; Professor M.J.H. Sterling for his help and guidance . Dr. M.R Irving and the O.C.E.P.S. research group at Durham for their assistance . Heather, my wife, for her encouragement, support and endless supply of coffee . My parents for the hours of proof reading and constructive comments and finally my grandparents for helping to finance the purchase of my word processor .



LIST OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF SYMBOLS

SYMBOL	MEANING
δ	Generator rotor angle
Δt	Iteration time step
E'	Voltage behind generator transient reactance
E_t	Voltage at generator terminals
F_o	System frequency (50 Hz)
F_{set}	Generator frequency set point
H	Generator inertia
I_t	Generator current
I_{inj}	Current injection into network
K	Generator governor gain
P_e	Generator electrical power produced
P_o	Generator output power
P_m	Generator input mechanical power
P_{set}	Generator power set point
$(P + jQ)$	Complex power
$(R_A + jX_D)$	Generator transient reactance
T_c	Turbine time constant
ω	Generator speed
V_p	Voltage at node p
$\dot{I}_p$	Injected current at node p
y_{pq}	admittance between nodes p and q

1 INTRODUCTION

Simulators are pieces of equipment which accurately model the behaviour of some form of physical system . They have been used in several engineering fields to allow control systems to be developed or for operators to learn about them : most often in cases where a mistake in the handling of the physical system would prove dangerous or costly . A prime example is the use of flight simulators in the training of commercial pilots : such simulators provide a realistic model of the cockpit of the aircraft in which the pilots sits . The view of the external terrain, which is modelled by computers, is projected onto screens outside the windows of the cockpit . The simulator relates the manipulation of the controls inside the cockpit to the effects on the view outside and allows the pilot to be subjected to emergency conditions, such as engine failure, without endangering lives or expensive equipment .

The physical system to be modelled for a power system simulator is made up of various items of equipment, linked together to supply power over a particular area . Electrical generators, driven by some form of prime mover such as a turbine, produce power which is transmitted to the consumers via lines and transformers . There are two distinct types of control on the system, operating on different time scales:

First the control which takes place locally and acts very quickly . For example, the mechanical governors and voltage regulators on generator sets . These attempt to regulate respectively

the speed and voltage produced by the generator to some set points . They begin to act as soon as the generator deviates from its set point . The set points themselves may come from some portion of the second type of control provided in the system .

The second type of control is the global operating scheme . This is the control of the system as a whole at regional or national level, rather than each individual item of equipment . The sorts of function performed centrally are the prediction of the amount of load in the near future and the scheduling of generators to produce the power required for the predicted load . The regional control must also reduce the likelihood of equipment failures, which can drastically affect the performance of the entire system, and cope with the effects of such failures if and when they occur .

In the U.K. the regional control system gathers its data from the network through a SCADA (Supervisory Control And Data Acquisition) computer system which is connected to many measuring devices and transducers throughout the system. The SCADA computer collects the measurements and presents them to the operators and to the control computers . Because of the amount of data that has to be collected it takes some time for the control centre to receive the data and act upon it: this means that the global control cannot act as quickly as the local control .

The OCEPS (Operational Control of Electric Power Systems) research group at Durham has been developing computer software to perform many of

the control functions performed in the control centres, both at regional and at national level . To test the algorithms produced a simulator is required which models the power system as accurately as possible .

Simulators have been produced in various forms for power system analysis for several years now . The basic problem is to model the dynamic components of the power system, such as the power station generators and the consumer loads, and compute the flow of power along the transmission lines connecting them. The simulator must be as controllable as the real system and respond to the controls in as similar a way as possible .

The uses and methods of simulation are varied and the accuracy and speed of the simulator depend on its intended purpose . The presentation of the calculated results and the input of changes to the system (referred to as the man-machine interface) also vary according to the use for which the simulator is designed . In general they are used much as flight simulators are used; to allow things to be done to the model which could prove costly and hazardous on the real system and to monitor the effects of those actions in detail .

1.1 SIMULATOR USES

Power system simulators can be used for a variety of different tasks, each task requiring the same end result - an accurate dynamic model of the power system . The presentation and time scale of the model vary, depending upon the type of task . Four main areas of importance can be

distinguished : energy management system development, transient studies, operator training and teaching . Each of these is described and discussed in turn below .

1.1.1 ENERGY MANAGEMENT SYSTEM DEVELOPMENT

The development of efficient power system control strategies is a field where large amounts of money can be saved by a small percentage decrease in operating costs . For instance the annual fuel cost of the C.E.G.B. is some £4000 million, so even a ½% reduction would amount to a saving of £20 million per year. Of course, the power system cannot be run safely on an economic least cost basis but account also has to be taken of the service to the consumer . Any control strategy must therefore balance the costs of running the system with the reliability and tolerance of the system to faults such as partial loss of generation, voltage reductions and so on .

A simulator is an essential tool for the development of a computerised control strategy - or energy management system (EMS) . It enables the computer programs to be tested under both normal running and emergency conditions without affecting the actual power system or the consumers it supplies . It is essential that the EMS should be thoroughly tested before it is used on the real system because mistakes and accidents can be costly to both the generating board and the consumer if the power supply fails . For this reason

the simulator must model as accurately as possible the response of the actual system to the controls applied by the EMS .

Several types of simulator have been produced as test beds using both digital (M. Rafian et al. [1]) and analog (R. Joetten et al. [2]) technology. To test the latest computer control algorithms the simulator should be as realistic a model of the system as possible, providing real time simulation and also representing the short term transients . However, the data produced by the simulator should correspond to the rates achieved by conventional SCADA systems so that the algorithms are not provided with more information than would normally be available . The possibility of incorrect measurements due to noise or failure of transducers should also be incorporated so that data validation and state estimation routines can be thoroughly tested .

The simulator for this type of use is accessed almost entirely by other programs and the need for any man-machine interface is minimal: all actions by operators would normally come through a SCADA program with its own interface . The only actions which need to go directly to the simulator are for debugging, which requires no more than a simple monitor with access to variables and the ability to run the programs step by step . All other instructions involve the passing of commands from the SCADA to the simulator and the returning of the simulated values . If the SCADA and the simulator are running in the same machine this can be done via a global memory area; if they are in separate machines a simple serial or parallel data line is all that is needed.

1.1.2 TRANSIENT STUDIES

The next area in which dynamic simulation is useful is that of examining the effects of transients in a system . Most SCADA systems on power systems present new data to the operator at relatively long time intervals (up to 30 seconds) and any short term effects are not seen at all . This does not matter to the operator because the effects are short term and he cannot do anything on so short a time scale . However it is useful to know what transients occur in a system when, for example, a circuit breaker is opened . For this type of information to be gathered, tests can be made on the real system or on a simulator . The simulator must have extremely accurate models of all the power system components and have a short enough time step to catch the transients which are being studied . However there is normally no need for the simulator to run in real time because there is no human interaction required.

Transient studies are useful in the design and tuning of the short time scale control which is applied locally to elements of the power system . The governor on a turbine can be altered to make the generator less likely to go unstable when equipment failures occur, but the tests on different governor parameters must be carried out with the generator connected to the rest of the system . This is because the response of the other generators and loads in the system may alter the response of the machine under test .

The man-machine interface for transient stability analysis consists of the ability to define the set of events to take place and

then, once the simulator has run, to view all the resulting data in a convenient form such as trend graphs . It is important that the tests should be repeatable so that the effects of altering of certain elements in the system can be seen clearly .

1.1.3 OPERATOR TRAINING

With the high reliability of modern power system components the occurrence of major faults such as islanding (the power system dividing into discrete parts) and large generation failures is rare . However it is precisely at these times that a well-trained operator may be able to play a crucial part in preventing or reducing the catastrophic effects such failures can have . Some post-fault studies have shown that the response of operators to serious system faults was not ideal, and there is accordingly a need for improved facilities for training operators to deal with major faults . Since the mid 1970's there has been an increase in the interest in and amount of work on operator (or dispatcher) training simulators [3-9] . This is due to three main factors:

a) Rising fuel costs have increased the amount of money to be saved by running the system closer to its operational limits and, in the case of regions with more than one generating authority, the exchange of power between authorities has become higher, pushing line flows close to limits and making unexpected outages of lines more important .

b) Environmental issues have forced new generation plant to be sited away from urban centres and hence away from the load . This again adds to the problem of high power flows along transmission lines .

c) The introduction of new types of equipment, such as pumped storage facilities, require the operator to be trained in the abilities and limitations of such equipment .

An operator training simulator must provide the trainee with an exact replica of the environment in which he is going to work. This can be achieved by building a control room and driving all the instrumentation and displays from the simulator instead of from the system itself . In this case the simulator has to have the same characteristics as the EMS development simulator so that the operator is given exactly the same amount of data as in a real control room in real time . As an alternative the dispatcher training simulator is run on the back-up computers in a real control room to provide the correct environment . This solution also saves the expense of building a replica control room and utilises the computing power available more fully . However, care has to be taken in such a system to ensure that a trainee cannot accidentally influence the real power system and that the back-up computer is always available for instant use should the primary control computer fail .

The operator also needs access to all the tools he would normally have at his disposal in the energy management system, such

as security analysis and generator rescheduling . Using such a system the operator can be trained in the use of the system and the software, starting with the normal operation of the plant, and then in dealing with major faults which can be introduced as frequently as the trainer wishes, giving the operator an amount of experience almost impossible to achieve without a simulator .

1.1.4 TEACHING

Using a simulator for teaching purposes is really an extension of the operator training problem . In teaching institutions such as polytechnics and universities where elements of power system theory are taught, practical work to back up the teaching is difficult because of the diversity of problems to be looked at and the expense of providing a small-scale low-voltage power system to be used as a model. However a simulator with a suitable man-machine interface can be used to give students a feel for the dynamics of the system and how changes in topology (network configuration) and system parameters can affect the running of the system. The simulator should give as much relevant information to the student as possible, even if this means that it no longer runs in real time, so that maximum benefit is gained from the time spent on it.

The man-machine interface for this type of use must allow the students to display all parameters of the system in an easily interpreted form, and to repeat the same scenario many times so that

all the effects of specific actions may be viewed . There should also be provision for the teaching staff to interact with the simulator at the same time so that parameters may be changed and the results viewed without having to restart the simulator with a new scenario of events to occur .

1.2 METHODS OF SIMULATION

There are three main types of simulator available . Each has its advantages and can be used for different types of problem:

1.2.1 ANALOG AND HYBRID SIMULATORS

The first power system simulators were analog machines using electrical components to model the various parts of the system . These simulators could run in real time because the models operated in parallel . However, with the advent of digital computers it became easier to model parts of the system and perform some of the control and data collection using digital hardware . These digital/analog machines are called hybrid simulators and are still used because of their faster than real time capabilities .

The digital part of the simulator is normally interfaced to the analog via a series of digital/analog and analog/digital converters which allow the computer to vary certain parameters and collect the

data as a normal SCADA system would . There are two main problems with both analog and hybrid simulators:

a) Even when the simulator is constructed out of modules, each module representing one type of component in the power system, the system topology is very inflexible . The hybrid simulator is slightly better than the analog, but if a new generator needs to be added to the system then another module has to be inserted by hand . If a completely different power system has to be simulated the simulator must be rewired in its entirety.

b) The cost of producing the components is high and they can be quite large . As a result the simulator for a large system can be bulky and also extremely expensive . For example the simulator developed by R. Joetten et al. [2] has modules on circuit boards 100 mm by 160 mm and it requires 12 such boards to represent a bank of three single phase, two winding transformers.

However for power authorities with small systems, hybrid simulators can be useful because the system to be modelled is reasonably static and, by using a base frequency higher than the system frequency, faster than real time simulation is possible for the study of long term dynamics of the system (G.E. Ott et al. [10]).

1.2.2 LOAD FLOW TYPE SIMULATORS

The most common type of digital simulator first uses a load flow approach to solve the network equations and then solves the dynamic equations such as the generator and load models using the initial nodal voltages and powers produced by the load flow (A. Keyhani [11]) . This technique is widely used for three main reasons:

a) First, because load flow calculations have been an important analysis tool for a long time . There are many well developed methods for calculating the load flow which are both fast and accurate . Methods such as the fast decoupled load flow (B. Stott and O. Alsac [12]) and the Newton load flow (W.F. Tinney and C.E. Hart [13]) can easily be adapted to work as the network solution part of a simulator .

b) Secondly, because the generator and load models are separate from the network equations they can be varied in complexity and accuracy depending on the type of work for which the simulator is to be used .

c) Thirdly, the load flow method gives fast computation of results, especially when the system is in a steady state condition, and it is thus easier to develop a simulator to run in real time .

Another time-saving feature of this technique is the fact that the load flow element need only be run once every few seconds while the dynamic model elements run far more frequently to give a more accurate simulation of the transient state of the system .

1.2.3 COMBINED DYNAMIC/ALGEBRAIC SIMULATORS

Combining the dynamic and algebraic equations of the network into a single algorithm and solving them simultaneously also provides a good method of simulation (M. Rafian et al. [1], L. Elder and M.J. Metcalfe [14]) . It can take longer than the previous method but has several advantages:

a) The method is highly accurate and stable: even if the system is split into islands by a line outage the simulator keeps going and automatically calculates the results for as many islands as are formed.

b) The load flow method always has the results of the dynamic and algebraic equations out of step by one integration whereas the combined method is tightly coupled and there is nothing out of step .

c) Although the combined method takes longer than the load flow method the correct choice of models for the system

components can still allow the simulator to produce results in real time .

The combined set of equations can also be split into those which require frequent calculation and those concerned with longer term dynamics . By removing the long term equations from the fast iteration the simulation can be speeded up .

1.3 SIMULATOR HARDWARE

Many forms of hardware are used for power system simulation, ranging from the circuit board modules used in the hybrid and analog simulators to the various forms of digital hardware grouped under the heading 'vector processors'. H.H. Happ et al. [15] examined the possibilities of the future technology while more recently D.M. Detig [16] and M. Takatoo et al. [17] have looked at the effects of vector processors on power system applications . Processors are often classified into groups according to their capabilities as follows:

a) SISD - Single Instruction Single Data:- Machines which execute one instruction at a time on one piece of data .

b) SIMD - Single Instruction Multiple Data:- Machines which execute one instruction at a time on a set of pieces of data .

c) MIMD - Multiple Instruction Multiple Data:- Machines which execute a set of instructions on a set of pieces of data simultaneously .

All the forms of hardware have their pros and cons . Those of the analog and hybrid hardware have already been mentioned: those of the digital machines available are as follows:

1.3.1 MAINFRAME AND SUPERCOMPUTERS

Mainframes such as the IBM 370 and supercomputers like the CRAY-1 are digital computers with large memory areas and extremely high computation speeds, ideal for large 'number crunching' operations like simulation problems . Unfortunately they are also extremely expensive and can be afforded only by large organisations and computer bureaux . Access to such machines is often possible but, for economic reasons, only in a time-sharing environment where a large number of users have the computer time divided between them . So a simulation task might run in real time if no one else was using the system but, under normal usage, the timing of the task would be much slower and entirely dependent on the load put upon the machine by other users .

Supercomputers often 'pipeline' instructions, that is they split each instruction into its component parts; fetch from memory, arithmetic operation and then store in memory; and can fetch the next

piece of data while the last one is in the arithmetic stages, which means that a far higher processing speed is obtained . Several pipelines often run together in step performing the same instructions on many pieces of data (SIMD) .

1.3.2 MIDI AND MINICOMPUTERS

Coming down the price scale in digital computing hardware are the midi and mini computers . These are more in the price range of large research groups . Naturally the power of these machines cannot compare to that of the supercomputers and the size of the memory available is smaller, but they do give a viable way of simulating power systems in real time, even if the systems that can be simulated in real time are restricted in size due to the lower computing power . These machines normally only operate as SISD machines and although some degree of pipelining may be possible there is normally only one pipeline in the machine .

One advantage the larger computers do have is that the development and operation of software can all be performed on one machine . Support for minicomputers in terms of software and hardware tools available is good and, with high level languages as standard, software written for one machine can easily be transferred onto a different machine .

1.3.3 ARRAY PROCESSORS

To improve the performance of minicomputers, without the expense of moving to a mainframe, array processors have been developed to provide execution rates far higher than available on minicomputers for certain types of operation . These machines require a minicomputer as a host and are used for performing arithmetic on vectors and arrays . The host computer normally loads the array processor by direct memory access (DMA) and lets the processor calculate the required result, for example a matrix multiplication, and then obtains the answer by DMA again .

The array processor is similar to the processing element in a mainframe, using a parallel, pipelined structure to give high computation rates for certain types of mathematical operations. An array processor is a form of MIMD machine but in a very restricted way; having for example two pipelines, one for addition/subtraction and the other for multiplication/division, and a processor dedicated to integer and indexing problems . D.M. Detig [16] suggests that array processors will not realise large improvements for power system applications . However recent developments such as cross compilers for FORTRAN allow array processors to be programmed in high level languages . This means that the effort required to put algorithms on array processors has been reduced and programmers do not need to know about the structure of the hardware or how to program it directly .

Using a cross compiler, the simulator currently used at Durham has been transferred onto a FPS 5205 array processor giving a speed-up of 3-4 times and, with further tailoring of the algorithm to suit array processor architecture, it is hoped to improve this .

1.3.4 MICROPROCESSORS AND MULTI MICROPROCESSORS

The growth in power and reduction in cost of microprocessors and their peripherals has led to a great deal of interest in using them for power system applications . The development of affordable microprocessors with high execution rates and 16 or 32 bit word lengths has meant that they can provide performance nearing that of minicomputers at a fraction of the cost . In addition, the software support for them (including compilers for many high level languages) means that code already developed can often be transferred onto microprocessors with a minimum of effort . To give one example of the high execution speeds possible, a benchmark run on a Motorolla microprocessor with floating point maths hardware took 1.5 times as long as a VAX 8600 which cost at least 30 times as much . While the structure of this benchmark certainly favoured the microprocessor it shows that for some types of application their value for money is excellent .

A natural step forward with such cheap processing power is to try to run several processors simultaneously on a problem . This configuration enables different operations to be carried out on

various pieces of data at any one time (MIMD) . Several studies have been made on the use of these multi microprocessor systems in power system analysis . H.H. Happ [18], J. Fong et al. [19] and F.M. Brasch et al. [20] looked at their application to power system studies in general, while W.L. Hatcher et al. [21], R. Lopez-Lopez [22], S.N. Talukdur et al. [23] and L. Dale et al. [24] looked at transient stability studies and simulation . Many other researchers have studied the general use of multi microprocessors, including H. Mukai [25], S.H. Fuller et al. [26] and A.K. Jones et al. [27] . Work on modelling their effects was undertaken by J. Grosser and S.N. Talukdur [28] who produced models to estimate the run time of algorithms when put onto a MIMD machine .

The main difficulty with the use of multiprocessor systems is that the problem has to be split into parts which can be solved simultaneously . With most computing being a sequential set of instructions this is not always easy . Some problems break down into parts quite readily while others simply cannot be computed in parallel and so no gain can be made by using more than one processor.

Several systems have been built, or designed, to utilise this type of architecture, the most widely known being the CM* at Carnegie-Mellon University on which several of the above papers are based . However any system developed as a general purpose system will not give optimum performance for a specific task and, with the low price and relative ease of construction, it may well prove

advantageous to design the hardware around the software if it is just to be used for one specific task.

1.4 MULTIPROCESSOR SIMULATION

The research group at Durham has been involved in the field of power system research for over 18 years and recently some third year undergraduates looked into the possibility of producing a power system simulator using a multi microprocessor system .First A. Perry [29] in 1982 and then J. Thomson [30] in 1983 carried out work which showed that it should be possible to produce a simulator which would run on hardware far cheaper than a minicomputer .

For any problem to be solved efficiently using a multiprocessor there are three requirements which have to be met by the software :

a) First, the problem must be split into parts which can be computed in parallel on the separate processors . Some problems are so non linear that this is impossible but for many problems at least part of the solution can be split up and performed in parallel .

b) Secondly, the amount of processing that has to be done centrally by a single processor must be minimised . The main advantage of a parallel processor is the speed of computation and any computing carried out centrally does not take advantage of this .

c) Thirdly, the time spent in communicating between the processors must be minimised . There is a finite limit on the speed of data transfer and this can be one of the major bottlenecks in any multiprocessor system . Not only is the amount of data to be transferred important but the time spent on setting up each transfer can be significant . It is, therefore, preferable to have a few transfers with large amounts of data than to have many small transfers .

Both Perry and Thomson used the TMS 9995 microprocessor for their studies . However with the advent of chips with 32 bit data structures it was decided to change to a more powerful processor . The main requirements for the processor for the multiprocessor simulator were those of processing power and support, both in hardware and software .

Two types of processor were looked at in detail, the Motorolla M68000 family and the Inmos Transputer . The processor that was finally decided upon was the M68000 for the following reasons:

1.4.1 HARDWARE SUPPORT

Hardware support is available in two forms . First the availability of a development system on which programs can be tested and debugged before implementing them on the actual hardware . Secondly the availability of hardware compatible with the microprocessor out of which a workable system to run the software can be built .

The M68000 had both these, a UNIX based development system being available within the university while boards using the processor could be designed and manufactured by the university . Also the M68000 was the first of a family of processors and more powerful versions would be available as the project went on . The Transputer was still under development and systems which emulated it were only just becoming available . It was, however, specifically designed for use as a multi processor system and in the future may well become an extremely powerful tool in parallel systems for all types of applications .

Finally, in terms of hardware support, the processor had to have specialised hardware available to perform the mathematical operations for floating point numbers because realising them in software is far too time consuming . Motorolla were due to bring out their maths co-processor for use with the M68000 family .However, other floating point hardware is available which can be used with the Motorolla microprocessor, some of which give far better performance than the Motorolla co-processor .

1.4.2 SOFTWARE SUPPORT

The availability of high level languages such as FORTRAN and C greatly helps the development of programs and makes them easier both to read and to change . Also any programming done in a high level language can be transferred onto other machines with a minimum of

effort so that the software does not become obsolete when the hardware is out of date . However when developing hardware as well as software it is quite often necessary to program in assembler as well as the high level languages and the availability of development software to facilitate this again saves time .

The Transputer was designed to be programmed using OCCAM, a concurrent programming language for use with multiple microprocessors. The M68000, using the UNIX development system had access to some powerful development and debugging tools and a number of different languages, some of which were also available on the university's mini computers as well . Thus algorithms could be tested on either the development system or a mini computer before they were tried on the multiprocessor system itself .

1.5 RESEARCH OBJECTIVES

The main objective of this research was to develop a real time power system simulator for use by the research team at Durham in the development of an energy management system . This objective can be split into a number of separate tasks :

a) First, to design efficient software for the power system simulation, capable of being used on its own as a research simulator and, in conjunction with an EMS, as a dispatcher training and teaching simulator . Also the simulator should, with the use of short

time steps, be able to simulate the transients in the system for transient studies .

b) Secondly, to develop microprocessor hardware to best utilise the parallelism of the software developed and produce a working simulator which was easily expandable to model different sizes of power systems .

c) Thirdly, to examine the limitations of the simulator developed . Both in terms of the size of system that could be modelled using the hardware and the time step that could be achieved by using different numbers of processors in the hardware .

1.6 THESIS LAYOUT

The original work put forward in this thesis, which can be found discussed in detail in the conclusions, is twofold . First the production of power system simulator software for use on parallel processors, in particular a parallel algorithm for the solution of the set of algebraic network equations, which minimises both the data transfer between processors and the amount of computing to be done centrally . Secondly, the implementation of the software on hardware suitable for exploiting the parallelism of the algorithms and the timing of the resulting simulator for various sizes of network .

The thesis itself can be split into four discrete parts :

a) Chapters 2,3 and 4 deal with the methods of modelling the various power system components, Chapters 2 and 3 covering the generator, load and network models suitable for parallel computation while Chapter 4 looks at the theory finally used on the parallel processors .

b) Chapters 5 and 6 deal in detail with the simulator that has been developed, Chapter 5 containing the software details while Chapter 6 describes the simulator hardware .

c) Chapters 7 and 8 present the results obtained, both the timings of the algorithms and the results of certain test runs carried out, and the conclusions drawn from the research .

d) The final part contains the references used in the thesis and the appendices . Each appendix contains some material to expand on a certain part of the body of the thesis .

2 GENERATION AND LOAD MODELLING

The simulation problem can be split into two parts . Firstly, the solution of the equations relating to the dynamic parts of the system, such as the generators and loads . Secondly the use of the results obtained for the generation and load values to produce a set of values for the voltages and power flows around the network . This split between the dynamic and network equations means that different generator and load models and solution techniques can be used without affecting the network solution . The only connection between the network and the dynamic solutions are the parameters produced by one part for the other part to use . Thus the only restriction placed upon the dynamic models is that they utilise and produce the correct parameters to interface correctly with the network model .

This possibility of changing the dynamic models used by the simulator means that different uses to which the simulator may be put can be accommodated . For instance for transient studies the dynamic models have to be extremely accurate and the time step very short in order to show the full dynamic response of the system to any changes made . The simulator may not run in real time with this sort of model, but this is not as important as the production of highly accurate results . However for training purposes the transients, over which the operator has no real control, need not be as accurately simulated but the results must be produced in real time so that the operator is given the right amount of time in which to respond .

This separation of the dynamic and network equations means that any dynamic model and solution technique can be used . By producing the software in a modular fashion the interchange of dynamic models can be made extremely easy . Several papers have been written on the affects of various load and generator models on the results produced in simulators [31-37] . These can help to give a guide in the choice of what complexity of dynamic model to use for each of the different uses of the multiprocessor simulator .

2.1 GENERATING EQUIPMENT MODELS

The models for the generating equipment in the system can be split into five separate parts . These are:

1) The supply, such as the boiler or reactor in thermal power stations and the reservoir in hydro stations . The thermal stations supply section must include elements such as the fuel supply to the boiler .

2) The prime movers, either gas turbines, steam turbines or hydro turbines and the mechanical governors controlling their rotational speed .

3) The shaft which transmits the mechanical power between the turbines and the synchronous machine . This has torsional elasticity,

inertia and damping due to friction in the bearings etc. which affect the dynamics of the system .

4) The synchronous machine itself, which converts the mechanical power produced by the turbine into electrical power for transmission across the grid .

5) Finally the excitation system which alters the field winding voltage in the synchronous machine to control the voltage produced at the machine terminals .

The most complex and important section of the model is the synchronous machine itself . This is because of the speed at which electrical transients can occur and the complexity of the interaction of the electric fields within the machine . The turbine and mechanical governor systems have slower time constants while the supply system reacts very slowly to any changes in the system . The time constants in the supply are of the order of minutes rather than seconds and those in the turbine range between about 10 seconds to 200 milliseconds .

2.1.1 SUPPLY SYSTEM

The supply system for a typical coal fired station is shown in figure 2.1 . The coal is fed via conveyers to pulverisers, the coal dust is mixed with air and blown through fans into the boiler furnace, the boiler then produces the steam to drive the turbines .

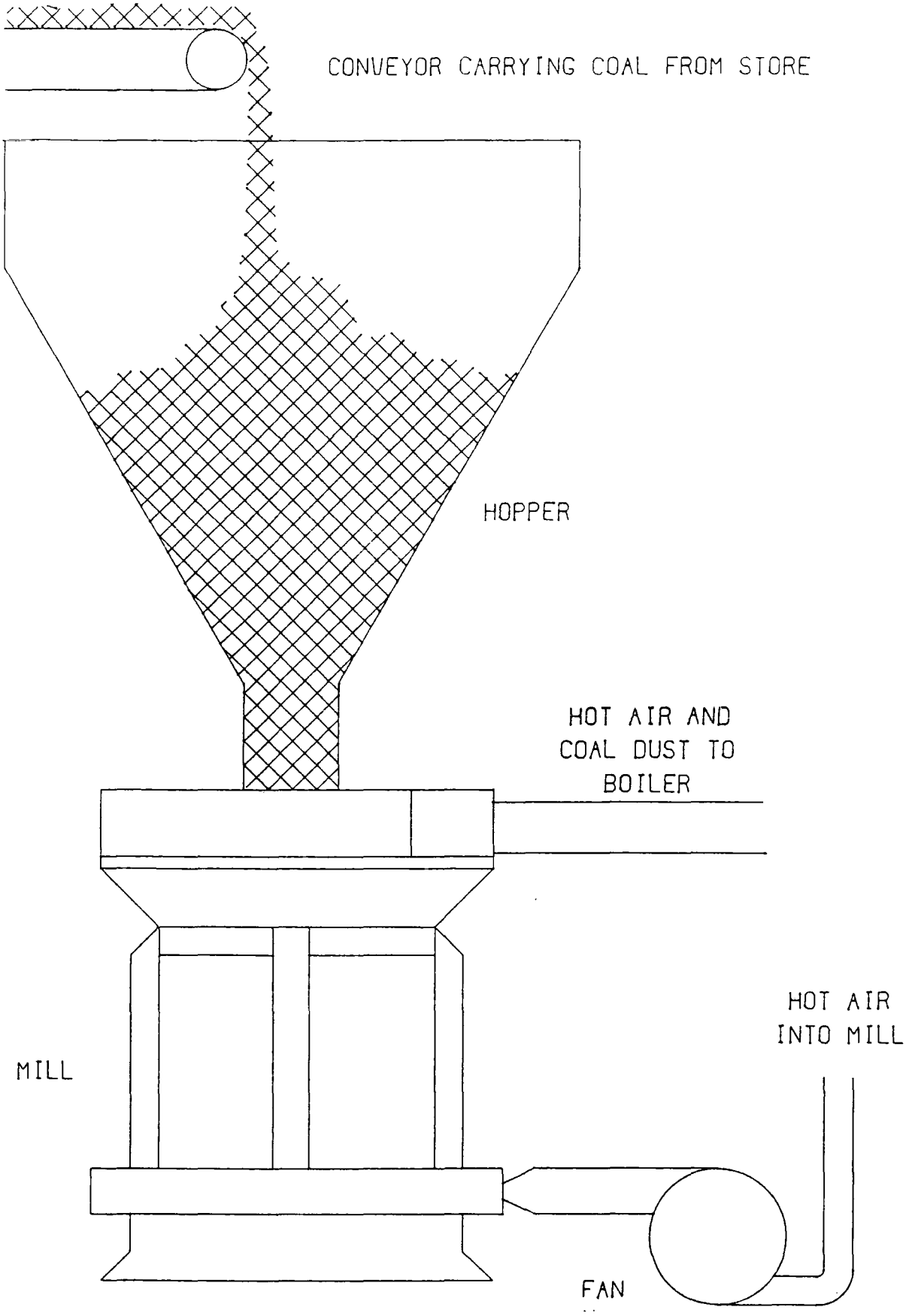


Figure 2.1) Coal supply system

There are several time constants involved in this sort of system: The amount of fuel processed by the pulverisers takes time to increase or decrease . The boiler heat output is slow to react to changes in the amount of fuel injected and there is often a large volume of steam reserve in the supply system so that changes in steam pressure occur slowly .

In a hydro station the simplicity of the supply system means that it has far fewer time constants and the power station can react far faster to changes in the demand . This means that large hydro stations such as the Dinorwig pump storage scheme in North Wales can be used to control the system frequency by going from standby to produce up to 1800 Meggawatts in only eleven seconds .

For thermal stations there are several different boiler models available, these often model the control on the boiler as several feedback loops using the boiler pressure, turbine steam flow and turbine speed . These, when modelled using the various time constants involved, give the boiler response to changes in turbine load . For the hydro supply the only time constant to be taken into account is that of the water supply to the turbine so the modelling is simplified greatly .

2.1.2 PRIME MOVER MODELLING

The prime movers are the machines which produce the rotation for the generators, this might be a series of steam turbines in a

thermal station as shown in figure 2.2 or a water turbine in a hydro station . For the thermal turbines there is a time constant before each of the turbines in the set corresponding to the steam chest, reheater or storage time . For example any change in boiler pressure first has to be passed through the steam chest before reaching the high pressure turbine, then to reach the intermediate pressure turbine the pressure change has to be relayed through the high pressure turbine and the reheater . A similar process occurs for the change to reach the low pressure turbines . Thus part of the pressure change affects the low pressure turbine a significant amount of time after affecting the high pressure turbine . These time lags should all be represented in the turbine model if highly accurate results are required from the simulator .

The other part of the generating equipment which is modelled with the turbine is the mechanical governor controlling the power input to the turbine in the form of steam input in thermal stations or water in hydro stations . This can be simply represented by a feedback loop with a single time constant representing the speed of response of the governor, a governor gain representing the sensitivity of the governor and upper and lower limits set on the amount of power input to the turbine . The rate of change of the input power can also be limited to model the speed of opening or closing of the steam valve controlled by the governor .

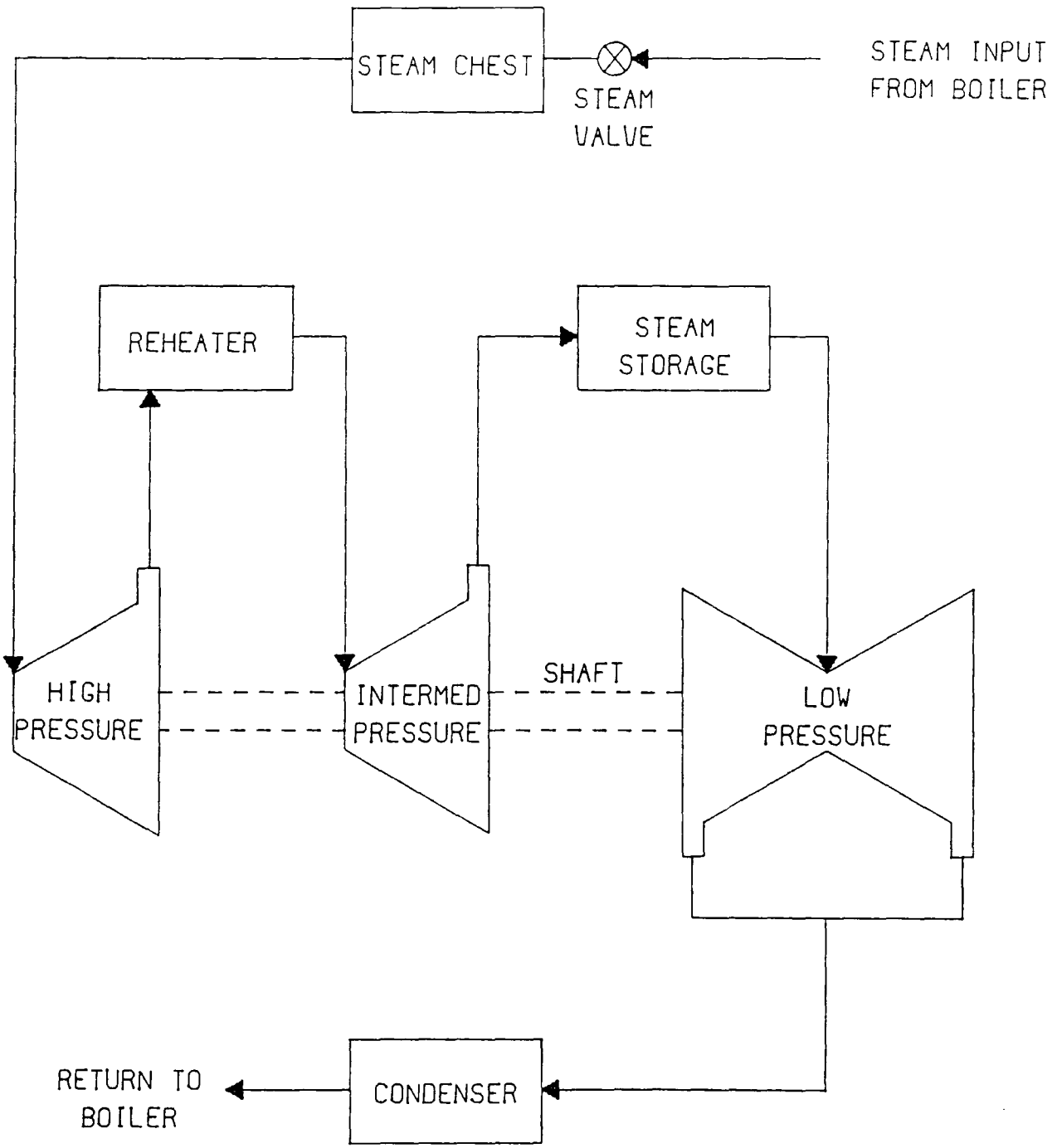


Figure 2.2) Prime mover string

2.1.3 SHAFT MODELLING

The shaft which connects the various turbines to the synchronous machines is usually an extremely large and heavy piece of equipment rotating at quite high speeds . The shaft has elastic properties inherent in its manufacture and damping due to the friction in the bearings and the drag due to the rotation of the shaft in the surrounding fluids . Thus the dynamics of the shaft itself can be extremely complex and can cause certain harmonic effects in the generating equipment .

For simulations in which the high speed oscillations of the shaft are not important the shaft can be modelled as a single lumped inertia with a damping term . For detailed transient studies the elasticity of the shaft has to be included to show the effects of resonance .

2.1.4 SYNCHRONOUS MACHINE MODELLING

For a balanced, three phase power system the equations governing the synchronous machine can be split into the direct and quadrature axis . This is done by using Parks equations . From these equations an approximate set of equations to represent the machine can be derived . The assumptions used to create this set of equations vary depending upon the use to which the simulator is to be put . For use in EMS development, operator training or teaching environments

the long term system dynamics are more important than the short term transients . Taking this into account a set of simplifying assumptions can be made to the synchronous machine model . These can give a representation as shown in figure 2.3 where:

E'	voltage behind generator impedance
E_t	voltage at generator terminals
X_D,X_Q	direct and quadrature axes reactances
R_a	generator resistance
I_t	generator current
I_D,I_Q	direct and quadrature components of generator current

The assumptions made to simplify the generator equations to this level are:

a) Neglect the modelling of all the harmonics of greater then second order . The amount of high order harmonics in a modern generating equipment is small so this is a reasonable simplification to the model .

b) Assume that the magnetic circuit in the synchronous machine performs in a totally linear way . For the modelling of slow transients this is a reasonable assumption. However, for transient studies needing high speed transient representation the saturation of the magnetic paths should be taken into account .

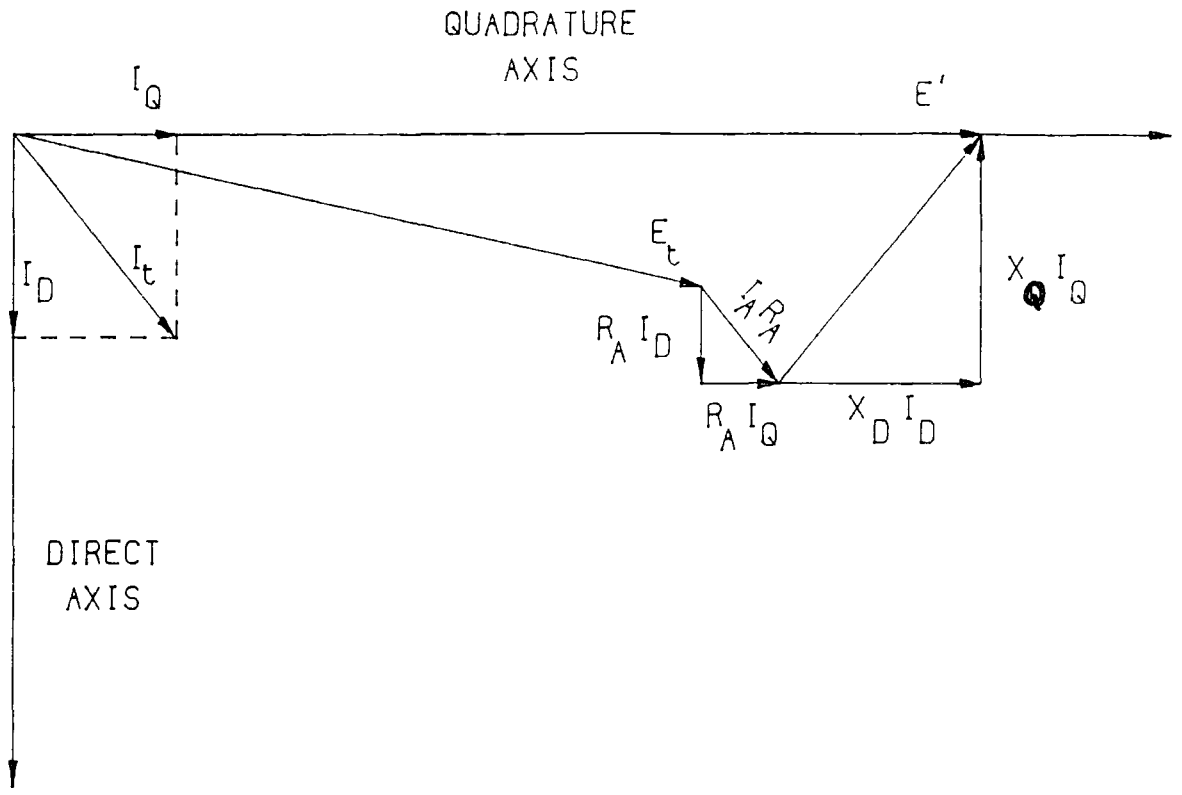


Figure 2.3) Approximate generator vector diagram

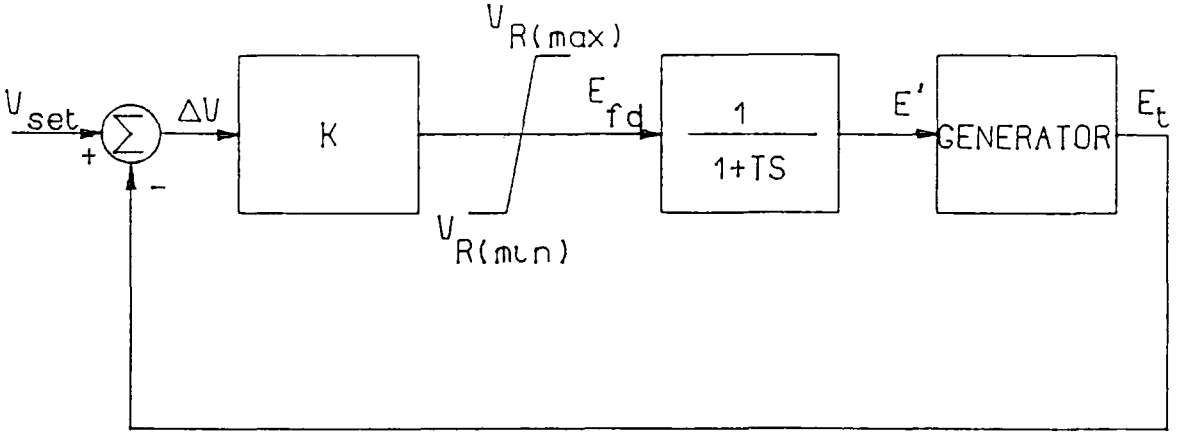


Figure 2.4) Excitation system block diagram

c) The independence of the generator's parameters from any frequency changes . Frequency dependence can be modelled if required but any increase in the model complexity will cause the simulation time to increase .

This model gives the following equations to govern the electrical characteristics of the synchronous machine.

$$E_t = E' - jX_D I_D - jX_Q I_Q - R_A I_A \qquad (2.1.1)$$

$$I_A = I_D + jI_Q \qquad (2.1.2)$$

If the further assumption that the direct and quadrature axes reactances are the same is made ($X_D = X_Q$) then equation (2.1.1) reduces to:

$$E_t = E' - jX_D I_D - jX_D I_Q - R_A I_A \qquad (2.1.3)$$

which, by incorporating equation (2.1.2),then reduces to:

$$E_t = E' - I_A * (R_A + jX_D) \qquad (2.1.4)$$

There are many assumptions that can be used to develop ways of obtaining the value for E', the voltage behind the machine impedance . The following equation couples the voltage with the field current :

$$E' = \omega * M * I_f \qquad (2.1.5)$$

where M is the mutual inductance between the field winding and the armature winding and ω is the rotor speed . This can be assumed to be constant the voltage becomes a simple function of rotor speed and

field current . For a far more simple model the value of E' can be fixed at a particular value governed by the initial conditions of the model . If any relation to the field current is used then a model of the excitation system used by the synchronous machine must be present to provide values of I_f .

2.1.5 EXCITATION SYSTEM MODELLING

On any synchronous machine there must be some form of excitation system to provide the voltage to the field winding . This voltage produces the magnetic flux in the generator which induces the electrical currents in the rotor windings .

The field voltage is often controlled by an Automatic Voltage Regulator (AVR) . This piece of equipment monitors the terminal voltage of the synchronous machine and varies the field voltage accordingly . This is, therefore, a simple feedback control system and has various time constants associated with it . Typically these time constants are short enough to be ignored for uses which do not require high speed transients to be modelled . A single time constant is included to model the time lag between the altering of the field voltage and the resultant change in the voltage behind the transient reactance . Also there are upper and lower limits placed upon the field voltage, this produces a model which can be represented by the block diagram shown in figure 2.4 . Where T is the time constant and K the gain in the feedback system .

2.2 LOAD MODELLING

The loads in the system also need to be modelled accurately to enable the simulator to produce satisfactory results . There are various types of load present in any network, from the simple lighting loads to the complex dynamic loads such as industrial motors . Several papers have been written on the effects of load modelling [34 - 37] and there are a series of models available for the different types of loads or for modelling a more general mixture of load .

2.2.1 GENERAL LOAD MODEL

One of the most commonly used load modelling techniques is to represent the way in which the magnitude of the load varies with changes in the supply voltage :

$$P + jQ = P_o * |V|^M + jQ_o * |V|^N \qquad (2.2.1)$$

where $P_o + jQ_o$ is the load power at per unit voltage and M and N are constants for the particular combination of loads being modelled . This model can be expanded further to include variations of the load due to fluctuations in the supply frequency :

$$P + jQ = P_o * (|V|^M + A(F - F_o)) + jQ_o * (|V|^N + B(F - F_o)) \qquad (2.2.2)$$

with A and B again being constants for the particular type of load, F is the system frequency and F_o the standard system frequency .

Three special cases of equation (2.2.1) are often used for simplifying the load model, these are: M=N=0 which gives a load with constant power characteristics, M=N=1 which gives constant current and M=N=2 representing constant impedance loads .

2.2.2 DYNAMIC LOAD MODEL

Another way of representing those loads in the system which are dynamic, such as industrial motors, is to use a similar model as that used for the synchronous machine in the generating station model. This is normally done by representing a group of motors as a single dynamic load with inertia and damping due to friction . The model acts in exactly the same way as in synchronous machine model but the input power from the turbine is replaced by a load on the motor (this load may vary with speed) . Also instead of producing electrical power and injecting it into the network the motor removes power from the system to satisfy the load .

3 NETWORK SOLUTION

The second part of the simulation problem is the solution of the network equations to determine the voltages at the nodes and the power flows along the lines . The network routine has to be performed twice for each time step: this is because the generator routine needs an intermediate network solution before it can calculate the final generator solution . The actual network values are then calculated by the network routine using the final generator solutions . The time taken for each network solution is, therefore, critical because there have to be two network solutions performed: thus the algorithm must minimise, as far as possible, both the amounts of data transfer needed and computing time taken .

There are several ways of solving the set of algebraic equations which govern the network characteristics and several assumptions which can be made to simplify the calculations . Some of these methods can readily be used on parallel processors while others are too non linear or involve too much data transfer to be split for parallel computation . Several methods were tried before one was developed which fully utilised the parallel processor architecture .

3.1. GAUSS METHOD

The Gauss method is a basic iterative method for solving a set of equations and it has several variations, such as the Gauss-Seidel method .

The standard method is simple to divide between a set of processors for parallel computation and the Gauss iteration also has the advantage of being very simple, both in mathematical technique and in programming required .

The equations representing the system can be represented in matrix form thus:

$$\begin{vmatrix} \\ Y \\ \\ \end{vmatrix} * \begin{vmatrix} V_1 \\ .. \\ .. \\ .. \\ V_n \end{vmatrix} = \begin{vmatrix} I_1 \\ .. \\ .. \\ .. \\ I_n \end{vmatrix} \qquad (3.1.1)$$

The exact formulation of this equation is covered in detail in the theory section, Chapter 4, of this thesis . It is simply a matrix version of Kirchhoff's current law which states that the sum of all currents entering a bus must be zero . The equation (3.1.1) is constructed using complex numbers, V_p is the voltage at node p while I_p is the injected current at that node . The Y matrix consists of elements representing the admittance and line charging of the lines connecting the nodes . Looking at a single row of equation (3.1.1) we obtain :

$$I_p = \sum_{q=1}^{q=n} (Y_{pq} * V_q) \qquad (3.1.2)$$

I_p is the current injected at node p due to any load or generation . If the total generation minus the total load at node p is $(P_p + jQ_p)$ then I_p can be written as :

$$I_p = \frac{(P_p + jQ_p)}{(V_p)^*} \qquad (3.1.3)$$

the * indicating the complex conjugate of a quantity . Combining equations (3.1.2) and (3.1.3) gives :

$$\frac{(P_p + jQ_p)}{(V_p)^*} = \sum_{q=1}^{q=n} (Y_{pq} * V_q) \qquad (3.1.4)$$

This can be rewritten in a form suitable for Gaussian iteration by taking the value V_p out of the summation on the right hand side (3.1.5). This equation is then executed iteratively for all nodes . The initial guesses for V are put into the right hand side and a new estimate calculated: the new estimate is then used on the right hand side until two consecutive sets of answers agree within a given tolerance . If V_p^k is the k^{th} estimate of the voltage at node p then the next estimate is calculated by :

$$V_p^{k+1} = \frac{(P_p + jQ_p)}{Y_{pp} * (V_p^k)^*} - \sum_{\substack{q=1 \\ q \neq p}}^{q=n} \frac{(Y_{pq} * V_q^k)}{Y_{pp}} \qquad (3.1.5)$$

The calculations can be speeded up further by using an acceleration factor on the calculated voltages . This simply involves increasing the size of the change in voltage at each iteration :

$$V^{k+1} = V^k + \alpha(V^{k+1} - V^k) \qquad (3.1.6)$$

where α is the acceleration factor . For load flow type calculations a value of around 1.2 or 1.3 has proved to give the best results . This whole algorithm can easily be divided into sections for parallel processing . Each processor is given a set of nodal voltages to calculate together with

a copy of the required parts of the Y matrix at the start of the simulation . The values of $(P_p + jQ_p)$ have to be updated each time step for those nodes with generators to incorporate the generated power calculated by the generator algorithm . Then at each iteration the latest values of V are passed to all processors and a new set calculated . This method has several advantages:

a) The Y matrix has relatively few elements and this sparsity helps to reduce both the memory requirements for the algorithm and the amount of non zero terms in the summation portion of equation (3.1.5) .

b) Any changes in the network such as the outage of a line require very little computation and the altering of only four of the elements in the Y matrix . Load changes affect only the $(P_p + jQ_p)$ value for a node, which is easily updated .

c) The computing can also be reduced by calculating the values of Y_{pq}/Y_{pp} at the start of the simulation as they remain constant throughout until affected by network changes .

d) The time for a single iteration is small and only one iteration is required when the system is in a steady state .

e) Although all the voltages have to be updated in each of the processors for each iteration the data transfer occurs in a single

block at the end of each iteration . This means that the time spent setting up the communication is small .

f) Finally, all the computing can be done in parallel . There is no need for any central calculation which would slow the algorithm down: only the communication with the host computer need be done by a central processor .

The Gauss method is well proven in its normal sequential form and the parallel form gives the same results . However, with some network configurations it fails to converge and simulation of the system becomes impossible by this method . The amount of data transferred between processors is quite large: all the nodal voltages have to be updated for each iteration . Also, when transients were introduced into the system, the number of iterations before convergence was achieved became large and the time taken by the algorithm, both in communication and computation, was far larger than the time step . This meant that real time simulation was not feasible .

As has been mentioned, there are several variations on the Gauss method . The most straight forward of these being the Gauss-Seidel in which, instead of waiting for a full set of voltages to be calculated before substituting them back in, the most recent voltage calculated is always used . For example when a new value of V_1 has been obtained it is used to help calculate V_2 and then both are used in V_3 and so on until V_n is calculated . From equation (3.1.5) this gives :

$$V_p^{k+1} = \frac{(P_p + jQ_p)}{Y_{pp} * (V_p^k)^*} - \sum_{q=1}^{q=p-1} \frac{(Y_{pq} * V_q^{k+1})}{Y_{pp}} - \sum_{q=p+1}^{q=n} \frac{(Y_{pq} * V_q^k)}{Y_{pp}} \qquad (3.1.7)$$

When this is implemented in a parallel form the data transfers between processors become spread throughout the algorithm instead of one block at the end of each iteration . This increases the overheads on the data transfers but reduces the number of iterations needed .

A compromise can be obtained by each processor using the latest values it has computed but ignoring those calculated by other processors until the start of the next iteration . This reduces the number of iterations without altering the data transfer, but does not have a large enough effect to enable real time simulation .

One possible solution to the problem of the time taken by the network solution would be to have a variable time step depending on the conditions . When the simulator needed several iterations to converge the time step could be lengthened to allow real time simulation to continue . However, this would loose one of the main attractions of having a short, fixed time step for the simulation ; that of modelling the transient response of the system .

3.2 NEWTON-RAPHSON METHOD

Another method of modelling the network is to use the Newton-Raphson load flow method . This method relates the active and reactive power mismatches in the system to the voltage angle and magnitude :

$$\begin{vmatrix} \Delta P \\ \Delta Q \end{vmatrix} = \begin{vmatrix} J_1 & J_2 \\ J_3 & J_4 \end{vmatrix} * \begin{vmatrix} \Delta\theta \\ \Delta V \end{vmatrix} \qquad (3.2.1)$$

The square matrix containing J_1,J_2,J_3 and J_4 is the Jacobian matrix of the system (W.F.Tinney and C.E. Hart [13]) . ΔP and ΔQ are the power mismatches in the system with the present estimates of voltage while $\Delta\theta$ and ΔV are the changes in voltage angle and magnitude to be made for the next estimate . This procedure is iterated until both $\Delta\theta$ and ΔV are smaller than a particular convergence value.

To divide the algorithm so it could be split into parts for parallel computation the power system was divided into areas . The solution of each area was then performed by a separate processor . The flow of power between the areas was represented by calculating the power flowing along the lines connecting the areas and including this power as an injection or drain on the relevant buses as shown in figures 3.1 and 3.2 . Thus a processor only needs to know the voltage of the bus at the far end of each of its inter area lines before the start of each iteration . The solution procedure is as follows :

1) Form the four submatrices J_1,J_2,J_3 and J_4 from the parameters of the system and combine these to give the complete Jacobian matrix as given in equation (3.2.1) .

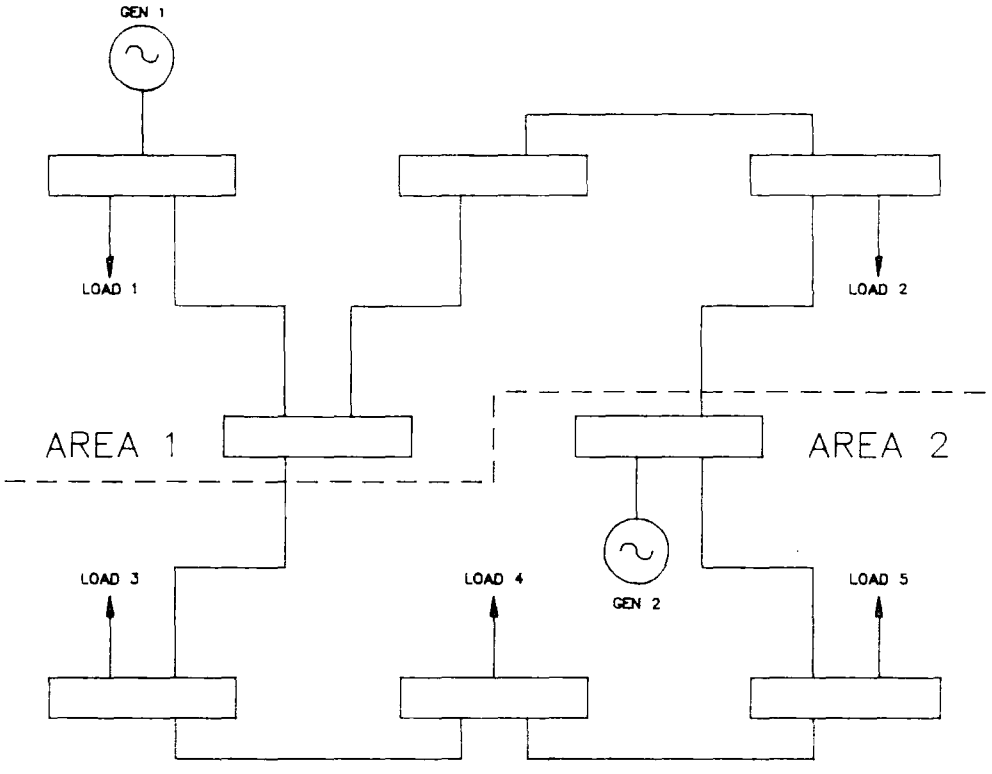


Figure 3.1) Example network divided into areas

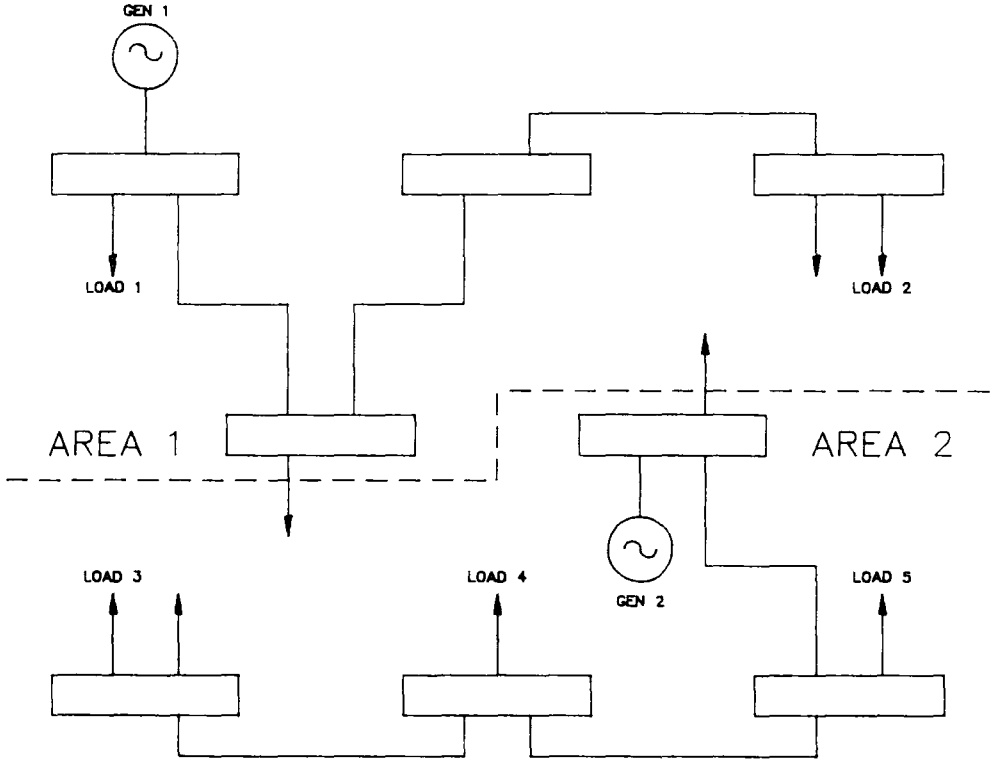


Figure 3.2) Connecting lines replaced by loads

2) Calculate the power mismatches ΔQ and ΔP at all buses including the dummy loads which represent the lines cut when the network is divided into areas .

3) Solve the equation (3.2.1) using Zollenkopf bi-factorisation [38] to produce $\Delta\theta$ and ΔV . Update the old values of V and θ using the calculated changes .

4) Repeat steps 2-3 until the values of ΔQ and ΔP are smaller than a given tolerance .

When a line outage occurs the Jacobian matrix changes . However, the Jacobian is only altered between time steps, never part way through a set of iterations . To make the calculation of the Jacobian easier the right hand column matrix is altered to contain $\Delta V/V$ instead of ΔV . This change does not affect the convergence or accuracy of the algorithm . This method has several advantages over the Gaussian method;

The data transfer between areas for each iteration is limited to the voltages of the buses at the ends of connecting lines . By choosing the areas to minimise the number of tie lines the data transfer is reduced . The transfers also all happen at once so both the data transferred and the overheads on setting up the communication are small .

The number of iterations for convergence during transient conditions is small and even though the calculation of the matrix

factors is quite lengthy this is only performed once per network solution .

The method does not have the convergence problems of the Gauss routine: therefore it is far more robust for use in simulation .

The major drawback with this method is the calculation of the relative phase angles between areas . Each processor calculates the voltages in its area with reference to some reference voltage . To calculate the power flow along the connecting lines the relative angles must be known . The use of a generator routine which produces voltages with respect to a constant 50 Hz all the areas effectively use the same reference voltage . However, if an area is left without generation because of an outage, problems can arise in the calculation and convergence .

One way of avoiding this situation is to split the system through the buses instead of the lines (figure 3.3) . This solves the angle problem because any one bus must have the same voltage and angle in all the areas in which it appears . The power flow in the lines between the areas is again represented as loads on the split buses . Unfortunately, the convergence of this method was found to be far worse than the initial method because the power transfer between areas became problematical .

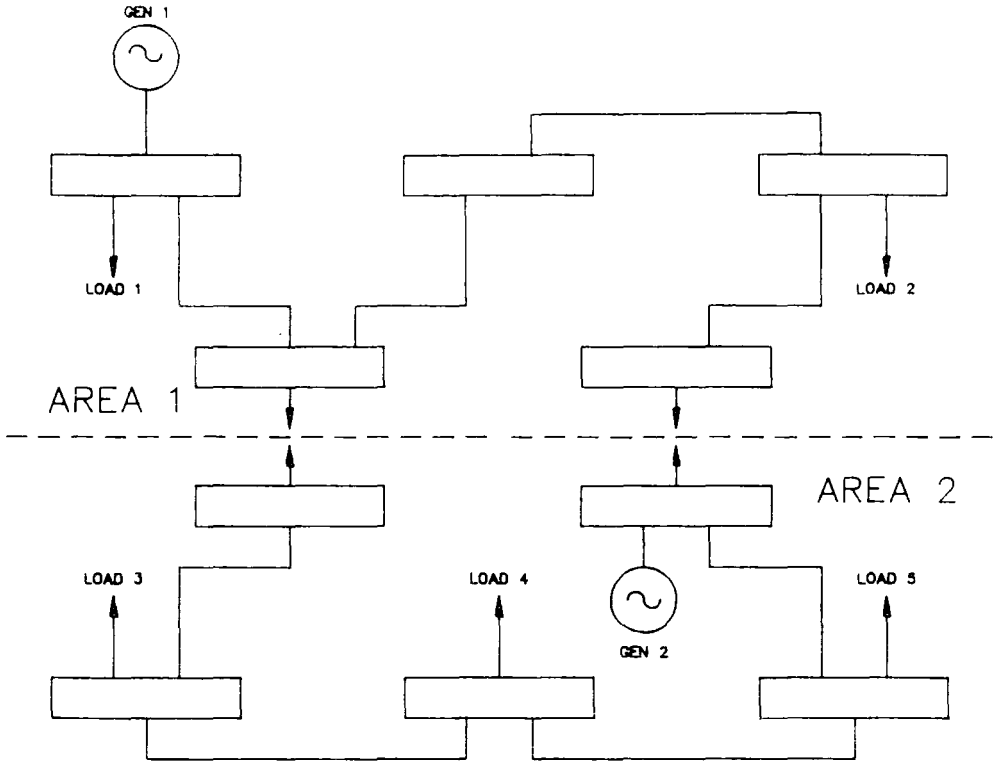


Figure 3.3) Network divided through buses

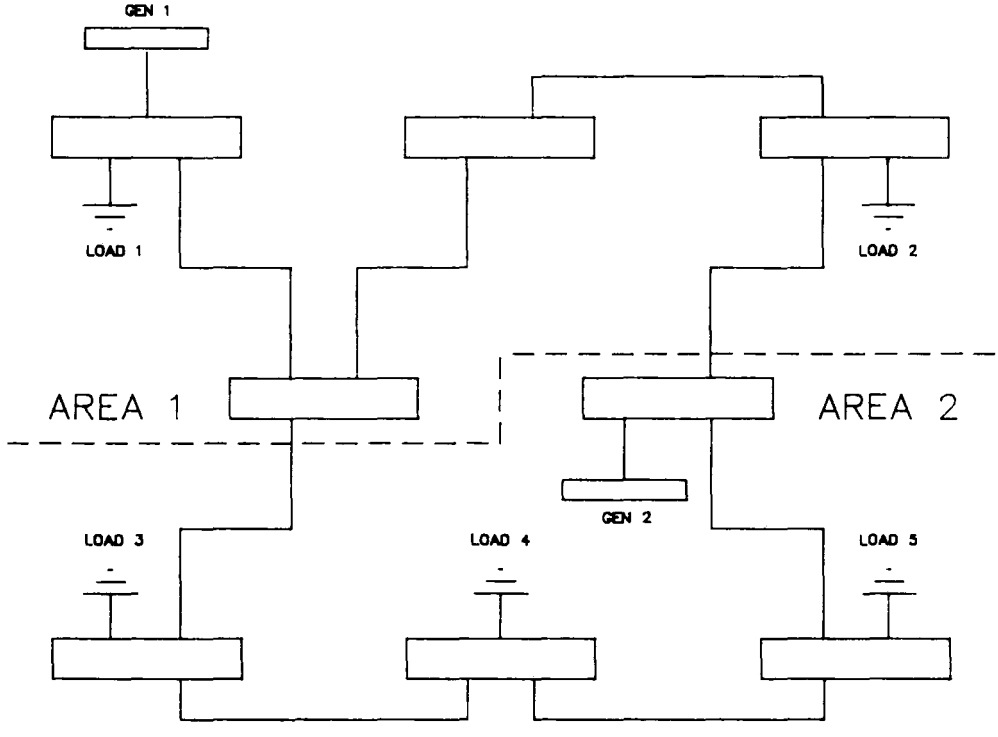


Figure 3.4) Network representation for direct method

3.3 DIAKOPTICS

Diakoptics is a method of solving parts of a problem as separate entities and then combining the partial solutions to help provide an overall solution . The name derives from the Greek *kopto* meaning to tear and so these methods are sometimes known simply as tearing or piecewise methods . The method was first put forward by G. Kron and has been applied to many fields of engineering including the solution of power system network equations .

This technique is obviously a candidate for parallel processing since it inherently divides the network into parts for separate solution . Several methods have been developed for solving load flow problems using diakoptics (H.H. Happ [39]) . The piecewise method looked at for the simulator was based upon the diakoptic version of the fast decoupled load flow (M. El-Marsafawi et al. [40]) .

Starting with the equation (3.2.1) this method neglects the small cross coupling between the real power and voltage magnitude (J_2 in the Jacobian) and between the reactive power and the voltage angle (J_3) . Further simplifications are then made (B. Stott and O. Alsac [12]) to obtain the following equations which have to be solved :

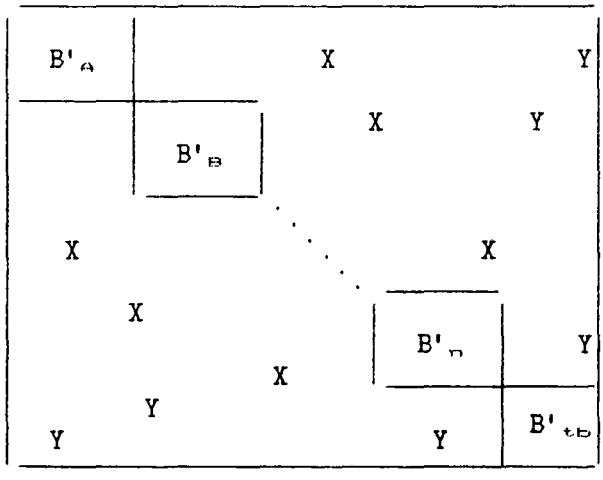
$$\begin{vmatrix} \Delta P/V \\ \cdot\cdot \\ \cdot\cdot \\ \cdot\cdot \\ \cdot\cdot \end{vmatrix} = \begin{vmatrix} \quad B' \quad \end{vmatrix} * \begin{vmatrix} \Delta\theta \\ \cdot\cdot \\ \cdot\cdot \\ \cdot\cdot \\ \cdot\cdot \end{vmatrix} \qquad (3.3.1)$$

and

$$\begin{vmatrix} \Delta Q/V \\ \cdot\cdot \\ \cdot\cdot \\ \cdot\cdot \\ \cdot\cdot \end{vmatrix} = \begin{vmatrix} \quad B'' \quad \end{vmatrix} * \begin{vmatrix} \Delta V \\ \cdot\cdot \\ \cdot\cdot \\ \cdot\cdot \\ \cdot\cdot \end{vmatrix} \qquad (3.3.2)$$

These equations relate the power mismatches at each node (ΔP is the active and ΔQ the reactive power mismatch) with the changes in voltage magnitude (ΔV) and voltage angle (Δθ) . B' is made up of the 1/x terms for the system (x being the line reactance) while B" is the negated bus susceptance matrix .

The system is split into areas and the B' and B" matrices are divided into those parts which can be used in parallel and those which have to be performed centrally . Both matrices can be split into parts corresponding to the areas of the system . Each area is given a temporary reference bus and the parameters for all the temporary buses are grouped together . The B' and B" matrices for the system can both be divided into two . First a set of submatrices which are relatively full and on the diagonal of the original matrix and secondly a very sparse set of off diagonal elements . Looking first at the B' matrix :



(3.3.3)

B'_n is the B' matrix for the area n of the torn system, B'_{tb} is the B' matrix for the temporary buses in the torn areas and the X elements are those representing the lines which join the areas and are cut when the system is torn . The Y elements represent the lines connected to each of the temporary buses in the system . Splitting the complete B' matrix into three parts to separate the calculations that can be done in parallel and those that have to be performed centrally :

$$B' = B'_{bdf} + B'_{ial} + B'_{ctb} \qquad (3.3.4)$$

B'_{bdf} is the block diagonal part of the B' matrix containing the submatrices B'_A to B'_{tb} . B'_{ial} contains the B' elements for the cut lines connecting the areas (the X elements in (3.3.3)), and B'_{ctb} contains the elements for the lines connected to the tie buses (the Y elements) . Substituting (3.3.4) into (3.3.1) :

$$\Delta P/V = (B'_{bdf} + B'_{ial} + B'_{ctb})\ \Delta\theta \qquad (3.3.5)$$

Which can be rewritten as :

$$(B'_{bdf}+B'_{ial})^{-1}\Delta P/V = (\ I + (B'_{bdf}+B'_{ial})^{-1}B'_{ctb})\ \Delta\theta \qquad (3.3.6)$$

Then the Householder matrix inversion formula [41] can be applied to the left hand side of equation :

$$(B'_{bdf}{}^{-1} - B'_{bdf}{}^{-1}(B'_{ial} + B'_{bdf}{}^{-1})^{-1}B'_{bdf}{}^{-1})\ \Delta P/V = (\ I + (B'_{bdf}+B'_{ial})^{-1}B'_{ctb})\ \Delta\theta \qquad (3.3.7)$$

If we let :

$$B'_{bdf}{}^{-1}\ \Delta P/V = \Delta\theta_1 \qquad (3.3.8)$$

and :

$$B'_{bdf}{}^{-1}(B'_{ial} + B'_{bdf}{}^{-1})^{-1}B'_{bdf}{}^{-1}\ \Delta P/V = \Delta\theta_2 \qquad (3.3.9)$$

Then :

$$\Delta\theta_1 - \Delta\theta_2 = (\ I + (B'_{bdf} + B'_{ial})^{-1}B'_{ctb})\ \Delta\theta \qquad (3.3.10)$$

Substituting equation (3.3.8) into (3.3.9) so that $\Delta\theta_2$ is calculated from $\Delta\theta_1$:

$$B'_{bdf}{}^{-1}(B'_{ial} + B'_{bdf}{}^{-1})^{-1}\Delta\theta_1 = \Delta\theta_2 \qquad (3.3.11)$$

This means that $\Delta\theta_1$ can be calculated from equation (3.3.8) and then modified by (3.3.11) to give $\Delta\theta_2$. These can then be used in (3.3.10) to give the actual angle change $\Delta\theta$.

This process(equations (3.3.3) to (3.3.11)) can be repeated with the B" matrix to give equations similar to those above dealing with the reactive power Q and the voltage V instead of the real power P and angle θ as follows :

$$B''_{bdf}{}^{-1}\ \Delta Q/V = \Delta V_1 \qquad (3.3.12)$$

$$B''_{bdf}{}^{-1}(B''_{ial} + B''_{bdf}{}^{-1})^{-1}\Delta V_1 = \Delta V_2 \qquad (3.3.13)$$

$$\Delta V_1 - \Delta V_2 = (\ I + (B''_{bdf} + B''_{ial})^{-1}B''_{ctb})\ \Delta V \qquad (3.3.14)$$

The equations can then be split for use on a parallel processor system by giving each processor the B'_{bdf} and B''_{bdf} for an area and allowing the calculation of $\Delta\theta_1$ and ΔV_1 for each area to be carried out in

parallel . One of the processors deals with the temporary buses instead of an area using B'_{tb} and B''_{tb} .

The solution of the P-θ equations is done separately from the Q-V equations and they are iterated until both the P and Q mismatches are less than a certain tolerance . The solution process of all the equations is done by matrix bi-factorisation rather than by computing the inverses of the matrices and is as follows :

1) Calculate the flow along the cut lines from the previous voltages at the line ends and the line admittances .

2) Form the ΔP/V active power mismatch vector for each area representing the power flows along the cut lines as additional loads at the buses at each end of the cut line .

3) Solve equation (3.3.8) in all processors to produce values of $\Delta\theta_1$. These values are then passed to a central processor for the modification to $\Delta\theta_2$.

4) Solve equation (3.3.11) in the central processor to give a value for $\Delta\theta_2$ and then solve (3.3.10) to produce the final angle changes Δθ. (The matrices B'_{im} and B'_{ctb} are both very sparse and this reduces the amount of calculation needed in arriving at Δθ .)

5) Update the angles θ in the system by adding the angle change Δθ to the previous value .

6) Perform steps 1 to 4 using the equations utilising the B" matrices and equations (3.3.12) to (3.3.14) to produce changes in voltage magnitude rather than angle . Update the voltages by adding the voltage change ΔV to the previous value .

7) Steps 1 to 6 are repeated iteratively until convergence is achieved . Convergence is checked by ensuring that the largest values of ΔP and ΔQ are both below a given tolerance: if they are not then the procedure is repeated from step 1 .

The fast decoupled technique has several advantages compared to some of the other diakoptic algorithms and the Gauss and Newton-Raphson methods :

a) The sparsity of the matrices used reduces the amount of memory required for storage and saves execution time . By using bi-factorisation there are no matrix inversions to be performed: this saves both the calculation time needed to perform the inverse and the storage requirements of the inverse which is a full matrix rather than a sparse one . To save more space, the matrices containing the parameters of the cut lines can be broken down further into a rectangular tie line connection matrix and a smaller square matrix containing the tie line elements (Appendix 1) .

b) The results produced by this method are the same as those produced if the system is left in one piece and then solved . There is no penalty on either the accuracy or the convergence as a result

of splitting the system into areas . The timing of the split and whole system solutions does vary slightly, but this is not significant compared to the speed up achieved by using a parallel processor system for the split algorithm .

c) Unlike some methods there is no restriction on the selection of the lines to be cut or the temporary buses . The selection of these does, however, have an effect on the solution time . The less lines that are cut the faster the solution . Also the problem is simplified if the cut lines are not connected to any of the temporary buses . There is no restriction on the size of the network which can be solved in this way .

d) The inclusion of the B'_{tml} and B''_{tml} matrices ensures that the cut lines are modelled exactly in both the active and reactive power mismatch problem . This also means that there is no difficulty finding the relative angles of all the areas like that encountered in the Newton-Raphson method .

e) The bi-factorisation is only carried out at the beginning of the simulation and when there is a change made to the network, the factors are always changed before a network calculation starts, never part way through a calculation .

f) The communication between the processors is limited to the central processor passing the new values of V and θ to the slave processors and then the slaves passing back their values for ΔV_n and

$\Delta\theta_{rr}$. Thus there are only four blocks of data transfer for each iteration, which means the overheads on the communication lines are small .

g) Even when transients are imposed on the system the number of iterations required for convergence is small, thus the time difference between steady state and transient calculations is far smaller than for the Gaussian method .

The main drawback of this method is the amount of computing that has to be done centrally . There is no way to avoid performing steps 4 and 5 centrally without vastly increasing the inter processor communication required for the solution . Therefore as the size of the network to be modelled is increased the number of areas and cut lines also has to increase so that the simulation can still be performed in real time . This means that the time spent in calculating steps 4 and 5 of the algorithm increases until the network can no longer be solved quickly enough for real time simulation .

3.4 DIRECT METHOD

The method finally developed for the network solution is a direct method using the Y matrix as given by equation (3.1.1) but inverting it to provide a direct solution rather than an iterative one . In this method the generators are represented as voltages behind a transient reactance and

the loads as either constant impedances to ground, as shown in figure 3.4, or as injected currents .

The theory of this method is given in detail in the next chapter which describes the theory of all the algorithms used in the final version of the simulator .

4 SIMULATOR THEORY

The theory used for multiprocessor simulation has several differences from that used upon standard hardware . The multiprocessor is used because of the high computation rates that can be achieved; however, as with any form of hardware, the multiprocessor has its limitations . The best algorithms will have to be built taking the limitations and capabilities of the hardware into account to produce a hardware/software solution to the simulation problem which is as efficient as possible . Very few of the methods of simulation used on standard hardware have both the parallelism and low data transfer rates needed for successful implementation on any multiprocessor system .

The final simulator algorithm developed for the multiprocessor can be divided into a number of smaller tasks, each of which can be split to run on a set of parallel processors . The tasks run in series . All processors must finish the first task before any can start the second, but the communication between processors occurs only at the end of each task and is kept to a minimum . For example, all processors run the network algorithm simultaneously to calculate the voltages in the system . When the processors have finished the network calculations the voltages are broadcast to all the other processors before the generator algorithm is started by all processors .

By splitting the theory, and the software using it, into modules corresponding to the components of the power system, the theory can be changed to produce as accurate a model of the system components as is

required . The generator module, for instance, can be replaced with any other module which calculates the same parameter (voltage behind transient reactance or generator current) for output at the end of the module . Initially the models of the generators and loads used are quite simple but these can easily be expanded to simulate components such as automatic voltage regulators on the generators and more complex couplings between the load and the voltage and frequency . These enhanced models can then be inserted instead of the generator or load modules in use at present .

The models used in the simulator are presented here in the simple form: these are fast but do not give a completely accurate model of the components .

4.1 GENERATOR REPRESENTATION

The generator algorithm is based upon one given in Stagg and El-Abiad [42] . The electro-mechanical equations of the turbine and generator represent them as a shaft or flywheel with an inertia H as shown in figure 4.1 . The equations relate the input mechanical power P_e and the output electrical power P_m to produce values of the changes of rotational speed ω and angle δ with respect to time :

$$\frac{d\delta}{dt} = \omega - 2\pi F_o \qquad (4.1.1)$$

$$\frac{d\omega}{dt} = \frac{\pi F_o}{H} * (P_m - P_e) \qquad (4.1.2)$$

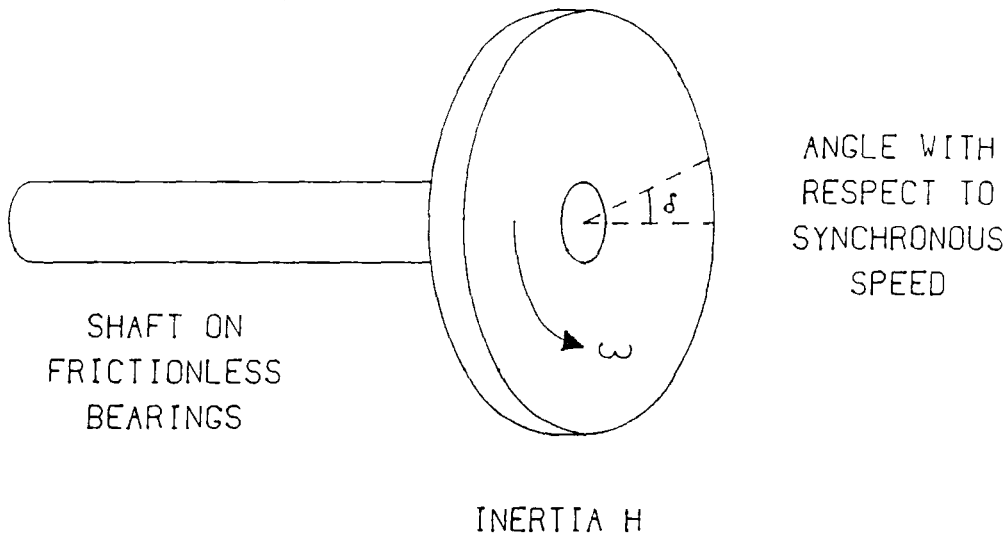


Figure 4.1) Turbine - generator representation

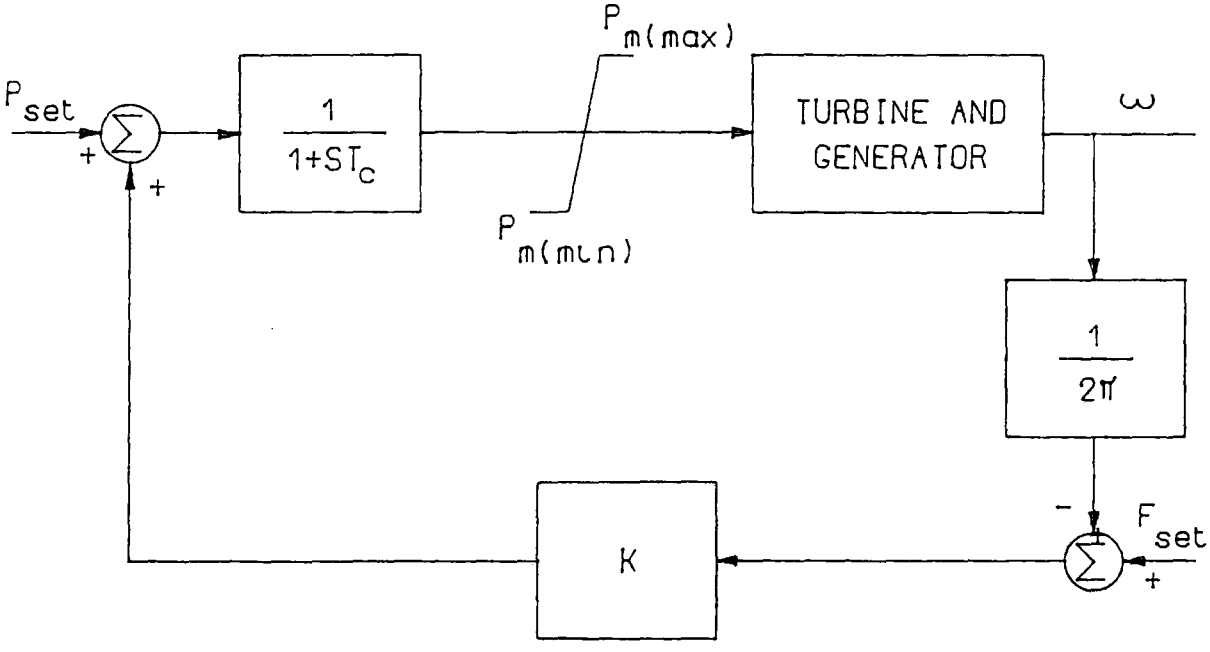


Figure 4.2) Governor model block diagram

where F_0 is the synchronous frequency in Hz . The Governor on the generator is then represented as shown in figure 4.2 . The mathematical equation which models this type of governor with a gain of K is :

$$\frac{dP_m}{dt} = \frac{P_{set} + K * (F_{set} - \omega/2\pi) - P_m}{T_c} \qquad (4.1.3)$$

Of the five components of a generating unit discussed in chapter 2 only three are included in this model . Equations (4.1.1) and (4.1.2) give the dynamics of the generator: this representation models the shaft as a single lumped inertia without any elastic properties . Equation (4.1.3) represents the turbine governor response to frequency and power deviations from the set points (F_{set} and P_{set}) and includes the turbine time constant T_c .

The boiler is not represented and it assumed that the turbine is always able to produce the necessary amount of mechanical power: there is also no representation of the excitation system . Both these components can be added to the set of equations to be solved if the use to which the simulator is to be put requires a more accurate model .

The initial values of the generator variables when the simulator is started are calculated from the values passed to the simulator from the host: these values are the results from a load flow calculation on the host computer . The initial values given to the generator program are P_g the generated power, E_t the voltage at the machine terminals and the generator impedance . All these values are complex . The generator impedance is its transient impedance $(R_A + jX_D)$.

The initial current at the terminals of the machine (I_t) is calculated from Ohm's law by:

$$I_t = \frac{P_g}{E_t^*} \qquad (4.1.4)$$

The initial voltage behind the transient reactance (E') is:

$$E' = E_t + ((R_A + jX_D) * I_t) \qquad (4.1.5)$$

The initial power angle δ is:

$$\delta = \tan^{-1} \frac{REAL(E')}{IMAG(E')} \qquad (4.1.6)$$

The initial air gap power (P_e) is:

$$P_e = REAL(P_g + | R_A*(I_t)^2 |) \qquad (4.1.7)$$

The initial mechanical power (P_m) is set equal to the air gap power:

$$P_m = P_e \qquad (4.1.8)$$

After the calculation of the initial values, the generator algorithm executes a modified Euler solution for each time step, obtaining from the master processor the latest value of the voltage at the bus to which it is connected and updating its power and frequency set points if necessary . The modified Euler routine calculates an estimate of δ, ω and P_m after time Δt based upon the previous values, then the terminal voltage is recalculated using those values . New estimates of δ, ω and P_m are then calculated and the average of the two results taken . The equations are as follows :

To calculate the current and electrical power :

$$I_t = \frac{(E' - E_t)}{(R_a + jX_d)} \qquad (4.1.9)$$

$$P_e = \text{REAL}(I_t * E'^*) \qquad (4.1.10)$$

Then to obtain the first estimates for δ, ω and P_m equations (4.1.1) to (4.1.3) are used to obtain values of $d\delta/dt$, $d\omega/dt$ and dP_m/dt which are then used to update the previous values of δ, ω and P_m :

$$\delta_1 = \delta + \frac{d\delta}{dt} * \Delta t \qquad (4.1.11)$$

$$\omega_1 = \omega + \frac{d\omega}{dt} * \Delta t \qquad (4.1.12)$$

$$P_{m1} = P_m + \frac{dP_m}{dt} * \Delta t \qquad (4.1.13)$$

The network routine is then run to recalculate the terminal voltage E_t using the initial estimates of δ, ω and P_m . Equations (4.1.9) and (4.1.10) are recalculated and then the final generator values are obtained thus :

$$\delta = \delta + \frac{(\frac{d\delta}{dt} + \omega_1 - 2\pi F_0) * \Delta t}{2} \qquad (4.1.14)$$

$$\omega = \omega + \frac{\frac{d\omega}{dt} + (\frac{\pi F_0 (P_{m1} - P_e)}{H}) * \Delta t}{2} \qquad (4.1.15)$$

$$P_m = P_m + \frac{\frac{dP_m}{dt} + (\frac{P_{set} + (K * (F_{set} - \frac{\omega_1}{2\pi})) - P_{m1}}{T_c}) * \Delta t}{2} \qquad (4.1.16)$$

These values are then used in producing the generator variable required by the network routine . This variable is either the value of the voltage behind the transient reactance E' or the machine output current I_t depending on the type of network solution . To calculate E' the magnitude calculated initially in (4.1.5) is kept constant but the voltage angle is δ: having calculated E' equation (4.1.9) is used to obtain I_t .

4.2 LOAD REPRESENTATION

Of the load models discussed in chapter 2 two were used in the simulator, either to model a load as a current injected into the network or as an impedance connecting the load bus to ground . The latter representation will be covered section 4.3 dealing with the network . The former can be split into two parts, the modelling of synchronous loads and the modelling of general loads .

4.2.1 SYNCHRONOUS LOAD MODEL

The modelling of dynamic loads such as synchronous motors is very similar to that of generators, the loads being represented by a set of differential equations such as (4.1.1) and (4.1.2). These equations are then solved by the application of a numerical technique such as the modified Euler used for the generator equations . The

result of the the numerical technique is a value for the current drawn from the network by the load .

The parameters such as load inertia are lumped parameters for all the synchronous load at the particular point . These parameters can only be calculated by tests being performed on the real network . The motor load does not need any form of governor representation because its speed is entirely dependent on the frequency of the supply and the load it is being used to drive .

4.2.2 GENERAL LOAD MODEL

Static loads, such as lighting and heating, are much simpler to model . Often they are regarded as fixed current, fixed impedance or fixed power or a combination of these three . They are modelled as a current drain from the network whose value is dependent upon both the voltage and frequency of the bus to which it is connected . The load power ($P_m + jQ_m$) at voltage (V_m) can be calculated from the nominal load power ($P_n + jQ_n$) at rated voltage (V_n) and frequency (F_O) by the following equation :

$$P_m + jQ_m = P_n * \left(\frac{|V_m|}{|V_n|}\right)^A + B(F_m - F_O) + jQ_n * \left(\frac{|V_m|}{|V_n|}\right)^C + D(F_m - F_O) \qquad (4.2.1)$$

where A,B,C and D are constants used to model the particular type of load required . Values for A,B,C and D can be ascertained for certain types of load but the constants can be determined accurately only by

tests on the real system as the component parts of a load vary widely .

Because of the length of time required to perform the raising to a power function (X^N) on the microprocessor the load was modelled in three parts using equation (4.2.1) . The three parts correspond to the constant power (A=C=0),constant current (A=C=1) and constant impedance (A=C=2) types of load with a variation added for frequency changes . This means the only power that has to be calculated is a square which can be calculated using multiplication .

The nominal load is split into three parts :

$$(P_n + jQ_n) = (P_0 + jQ_0) + (P_1 + jQ_1) + (P_2 + jQ_2) \qquad (4.2.2)$$

and the following summation is used to give the total load power :

$$P_a + jQ_a = \sum_{i=0}^{i=2} P_i * \left(\frac{|V_a|}{|V_n|}\right)^i + B_i(F_a - F_0) + jQ_i * \left(\frac{|V_a|}{|V_n|}\right)^i + D_i(F_a - F_0) \qquad (4.2.3)$$

The load model is shown diagrammatically in figure 4.3 . P_1 is the constant power part ($P_0 + jQ_0$), I_1 is the constant current part of the load ($P_1 + jQ_1$) and the impedance to ground produces ($P_2 + jQ_2$) . Each of the three parts of the load has its own values for the constants B and D to represent the rate of change of power with respect to frequency . Having calculated the power of the load the current injection for the load bus is then calculated by :

$$I_{inj} = - \frac{P_a + jQ_a}{V_a^*} \qquad (4.2.4)$$

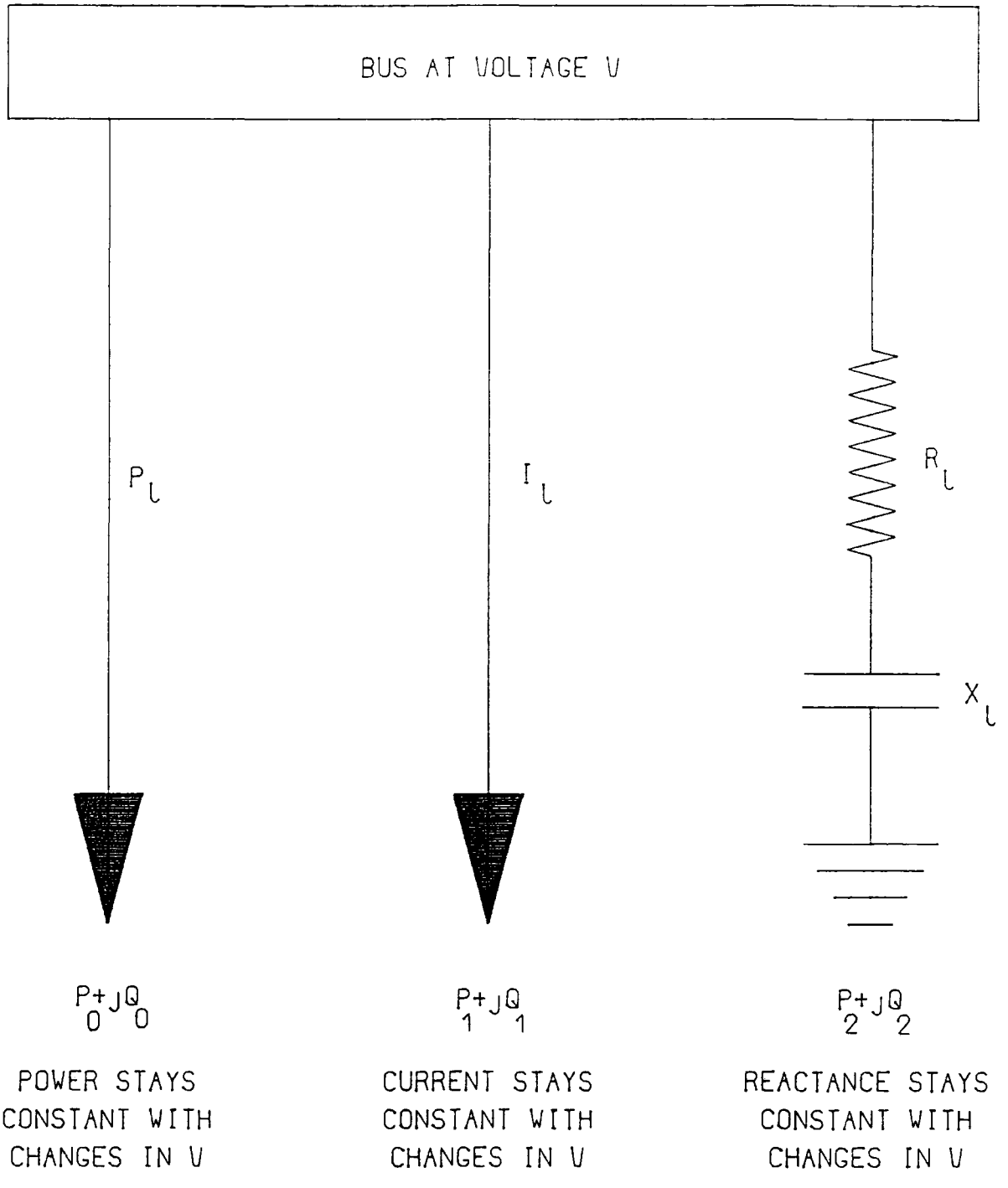


Figure 4.3) General load representation

This injection is then used by the network solution which calculates the voltages of the buses in the system .

This general load model is used more often than the synchronous load model because the loads at a bus are normally a mixture of different types . Equation (4.2.1) can be performed but the time taken for the routine using (4.2.1) is over twice as long as if (4.2.3) is used .

4.3 NETWORK REPRESENTATION

Figure 3.4 shows the way a network is modelled by the simulator with all the loads regarded as constant impedances and all the generators as voltages behind transient reactances . The generators are represented as buses connected to the rest of the system by a line with the generator's transient reactance . The voltages of these buses are obtained from the generator algorithms and are regarded as fixed for the duration of each time step .

The loads are represented as a line from the load bus to ground, the line impedance being calculated to give the required load power at rated voltage . This constant impedance representation is equivalent to the general load model (4.2.3) with all the load in (P_{z} + jQ_{z}) and no frequency variation .

The network itself is made up of a series of buses connected by lines and transformers . The lines are modelled as a Π model with a

resistance and reactance in series and line charging capacitances at each end of the line connecting the buses to ground . Transformers are represented as lines with reactance equal to the leakage reactance of the transformer . The buses at each end of a transformer connection have different per unit bases to account for the voltage change across the transformer . An extra bus (bus 0) is introduced as the ground bus, all line charging elements and constant impedance loads are lines connected to this bus .

From this representation a matrix of the system admittances, called Y is built up where the off diagonal elements are given by :

$$Y_{pq} = -y_{pq} \qquad (4.3.1)$$

y_{pq} being the admittance of the lines joining bus p and bus q . The diagonal elements are given by :

$$Y_{pp} = y_p + \sum_{q=0}^{q=n} y_{pq} \qquad (4.3.2)$$

y_p being the sum of the line charging to ground at bus p . This Y matrix can then be used in a form of Ohm's law to calculate the system voltages and injected currents . If V_p is the voltage at bus p and I_p is the injected current at bus p then, putting the voltages and currents in the form of column vectors we get :

$$\begin{vmatrix} \\ Y \\ \\ \end{vmatrix} * \begin{vmatrix} V_1 \\ .. \\ .. \\ .. \\ V_n \end{vmatrix} = \begin{vmatrix} I_1 \\ .. \\ .. \\ .. \\ I_n \end{vmatrix} \qquad (4.3.3)$$

The row and column corresponding to bus 0 (the ground bus) have been eliminated because the voltage is zero and we are not interested in the injected current: this also makes the Y matrix non singular so it may be inverted . If the buses are numbered so that those representing the generator voltages behind their transient reactances occur first then the Y matrix can easily be divided into four submatrices :

$$\begin{vmatrix} \\ Y \\ \\ \end{vmatrix} = \begin{vmatrix} A & C \\ B & D \end{vmatrix} \qquad (4.3.4)$$

If *ng* is the number of generators in the system and *nb* the number of buses then the matrix A is square (*ng*ng*), diagonal and contains the generator admittances . The matrices B (*nb*ng*) and C (*ng*nb*) contain the corresponding off-diagonal terms and the D (*nb*nb*) matrix contains the line, transformer and load representations, in the form of admittances, for the remainder of the system .

$$\begin{vmatrix} A & C \\ B & D \end{vmatrix} * \begin{vmatrix} V_k \\ \\ V_u \end{vmatrix} = \begin{vmatrix} I_u \\ \\ I_k \end{vmatrix} \qquad (4.3.5)$$

The voltage and current matrices are divided into two portions each: one portion whose values are known V_k (*ng*1*) and I_k (*nb*1*) and one whose values need to be calculated V_u (*nb*1*) and I_u (*ng*1*) . The known voltages are those calculated by the generator algorithms as the voltage behind the

generator's transient reactance, while the injected current at all buses without generators attached will be zero . Thus the equation can be rewritten by moving the known and unknown values of voltage and current onto separate sides :

$$\begin{bmatrix} A & \emptyset \\ -B & I \end{bmatrix} * \begin{bmatrix} V_k \\ I_k \end{bmatrix} = \begin{bmatrix} I & -C \\ \emptyset & D \end{bmatrix} * \begin{bmatrix} I_u \\ V_u \end{bmatrix} \qquad (4.3.6)$$

where $\emptyset$ is an empty matrix and I is the identity matrix . To obtain an equation with only the unknown column vector on the right hand side premultiply both sides by the inverse of the right hand square matrix to give :

$$\begin{bmatrix} I & CD^{-1} \\ \emptyset & D^{-1} \end{bmatrix} * \begin{bmatrix} A & \emptyset \\ -B & I \end{bmatrix} * \begin{bmatrix} V_k \\ I_k \end{bmatrix} = \begin{bmatrix} I_u \\ V_u \end{bmatrix} \qquad (4.3.7)$$

And hence, by performing the multiplication of the two square matrices in (4.3.7) :

$$\begin{bmatrix} A-CD^{-1}B & CD^{-1} \\ -D^{-1}B & D^{-1} \end{bmatrix} * \begin{bmatrix} V_k \\ I_k \end{bmatrix} = \begin{bmatrix} I_u \\ V_u \end{bmatrix} \qquad (4.3.8)$$

If all the loads in the system are represented by a constant impedance connected to ground then all the known injected currents are zero . Thus the I_k matrix can be replaced by a Ø matrix which means that a large proportion of the terms in the square matrix can be set to zero and the problem is simplified .

$$\begin{vmatrix} A-CD^{-1}B: & Ø \\ \cdots\cdots : \cdots\cdots \\ -D^{-1}B : & Ø \end{vmatrix} * \begin{vmatrix} V_k \\ \cdots \\ Ø \end{vmatrix} = \begin{vmatrix} I_u \\ \cdots \\ V_u \end{vmatrix} \qquad (4.3.9)$$

The network algorithm needs only calculate the unknown voltages since the injected currents are calculated by the generator algorithm . Hence all the network program has to do is perform the matrix multiplication :

$$\underset{nb*ng}{\begin{vmatrix} -D^{-1}B \end{vmatrix}} * \underset{ng*1}{\begin{vmatrix} V_k \end{vmatrix}} = \underset{nb*1}{\begin{vmatrix} V_u \end{vmatrix}} \qquad (4.3.10)$$

This multiplication is further simplified by the fact that the B matrix has only ng non zero elements and each column and row can have, at most, one non-zero element. This means that the B and D^{-1} matrices are held separately and their multiplication is done simultaneously with the equation (4.3.10) each time the network algorithm runs :

$$\underset{nb*nb}{\begin{vmatrix} -D^{-1} \end{vmatrix}} * \underset{nb*ng}{\begin{vmatrix} B \end{vmatrix}} * \underset{ng*1}{\begin{vmatrix} V_k \end{vmatrix}} = \underset{nb*1}{\begin{vmatrix} V_u \end{vmatrix}} \qquad (4.3.11)$$

The sparsity of B means that for each unknown voltage required only $2*ng$ complex multiplies and ng complex adds are needed . However this assumes that the inverted D matrix is always up to date, any topology changes or load variation causes the entire D^{-1} to change . The computing of equation (4.3.11) is very simple to split between processors: each processor is given a fixed set of bus voltages to calculate (v_{ba}), the corresponding part of the D^{-1} matrix (d^{-1}) and the B matrix .

$$\underset{nba*nb}{\left|\,-d^{-1}\,\right|} \;*\; \underset{nb*ng}{\left|\,B\,\right|} \;*\; \underset{ng*1}{\left|\,V_{bus}\,\right|} \;=\; \underset{nba*1}{\left|\,v_{ba}\,\right|} \qquad (4.3.12)$$

where nba is the number of buses to be calculated by the processor .

Thus each time the algorithm runs the most up to date values of V_{bus} are obtained from the master processor and equation (4.3.12) is performed and the results sent back to the master processor . This model has several advantages :

a) It is fast and non iterative so the execution times for the network and generator solutions are the same in both steady state and transient conditions .

b) The computation is performed entirely on the slave processors: the master is only required to communicate with the slaves and transfer the results to the host computer .

c) The B matrix is so sparse that it only contains ng non zero elements . Instead of the whole B matrix, each processor is given the

generator transient reactances and the bus numbers to which the generators are connected .

There are also three problems with the model used by equation (4.3.12) :

a) First, the loads are modelled as constant impedance loads because they are included as part of the network . This is accurate for certain types of load but not for others .

b) Secondly, to alter the load the inverted matrix must be altered . The technique works well for small systems, but when larger systems are simulated the overheads involved in altering the inverted matrix become large .

c) Finally, the inverted matrix is full rather than sparse: this means far more memory is needed than for other methods which can use sparse matrix techniques .

To overcome the first two problem the loads can be modelled as injected currents . This allows a load model such as equation (4.2.3) to be used and reduces the amount of change to the matrix inverse . This means that equation (4.3.8) must be solved in full: only those nodes in the system without generation or load have zero injected current so the computation for the network solution rises . However the method is still fast and non iterative . The parallel solution of this problem can be simplified if the generators and loads both produce a current injection

onto the bus to which they are connected . This can be done by calculating the value of the generator current (I_t) at the end of the generator calculations . The equation to be solved can then be written :

$$\begin{array}{c} nba*nb \\ \hline \\ \left| \; -d^{-1} \; \right| \end{array} * \begin{array}{c} nb*1 \\ \hline \\ \left| \; I_w \; \right| \end{array} = \begin{array}{c} nba*1 \\ \hline \\ \left| \; v_{..} \; \right| \end{array} \qquad (4.3.13)$$

Some of the injections will still be zero but the calculation is larger than that required for (4.3.12) . However with the inclusion of a variable time step it can easily run in real time .

The problem of the memory required by the method is far less important now than it has been historically for two reasons . The price of memory is now very low compared to the price of the processors and other system components, so large amounts of memory do not increase the price of the system dramatically . Also with 16 and 32 bit processors the amount of memory which can be addressed is large and a 32 bit address bus poses no practical limit on the amount of memory which can be used . The only real limit is set by the capacitance effects of the memory slowing down the signals on the bus and this does not significantly affect the system even when 1 or 2 megabytes of memory are used .

4.4 MATRIX ALTERATION ALGORITHM

The simplicity of the network routine is due mainly to the fact that it is assumed that an accurate version of the D^{-1} matrix is always available . When the simulator is started a copy of this matrix, the

inversion having been done on the host machine, is loaded into the simulator along with the system topology . However as soon as a line is removed or a constant impedance load altered the whole matrix changes and must be recalculated .

The inversion of the D matrix is extremely lengthy for anything but the smallest systems. It has to be performed once on the host machine before the simulator can be run, but this does not affect the simulator timings . However, because the changes to the D matrix involve a small number of elements the inverted matrix can be altered, rather than reinverted, to give the inverse of the new D matrix . Using the formula given by Householder [41] which calculates the inverse of a modified matrix using the inverse of the original :

$$(D + UXV^T)^{-1} = D^{-1} - D^{-1}UX(X+XV^TD^{-1}UX)^{-1}XV^TD^{-1} \qquad (4.4.1)$$

where U and V are rectangular and X is square and all the dimensions are correctly matched .

Because the ground bus is removed from the matrix the alteration of a load means that only one element of the D matrix is altered . A line outage alters four elements of the matrix, two in the column representing each of the connected buses . The line outage case is divided into two separate passes, one for each column affected by the change, so both the load and line cases can be handled by the special case when U and V are column vectors u and v, and the square matrix X is a scalar equal to 1 which gives:

$$(D + uv^T) = D^{-1} - (D^{-1}u)(v^TD^{-1})/(1 + v^TD^{-1}u) \qquad (4.4.2)$$

This is further simplified by the fact that u and v have only one non zero element for a load change and two for a line outage . If v has the required complex values in it then the non zero elements of u must have the value 1 . Using this method a line outage needs two passes through the routine but a load change needs only one pass.

The equation (4.4.2) is split between the processors so that each slave modifies the portion of the D^{-1} it contains for the calculation of the bus voltages allocated to it . A certain amount of the calculation has to be done on one slave and the results passed to all the other slaves for the modification of their own sections . The calculation required centrally is that of the divisor in equation (4.4.2) :

$$S = (1 + v^T D^{-1} u) \qquad (4.4.3)$$

Then the value of S and the column of D^{-1} which corresponds to the bus being altered are broadcast to all the slaves in the multiprocessor and equation (4.4.2) is completed in each slave .

The Householder routine is needed only when the network is changed . Thus if no change occurs a lot of time is wasted because the master processor waits for the correct time before starting the next time step. There are two ways to minimise the amount of time wasted in this way:

The Householder routine can be prevented from running every time step and only allowed to run every few time steps: this reduces the time wasted but does not eliminate it . This method also has the disadvantage that the simulator may not respond as quickly to control commands as the real system.

Secondly the time step used can be varied depending upon whether or not the Householder routine is run: if the slave processors receive no network change from the master processor they use a short time step . When one or more passes through the Householder routine are needed a longer interval is used for the next time step . Both time steps involve a period for changes to generator set points and separately modelled loads but these take an extremely short amount of time compared to the Householder routine . This means that very little time is wasted and the system responds more quickly to commands .

The second method does not have the disadvantages that a variable time step can have in iterative solution techniques . In an iterative technique the time step is increased when a large number of iterations are required to produce a solution: this occurs while the network is in a transient state . In the direct method the time step is only lengthened when a change occurs . The rest of the transient can be modelled at the shorter time step until another change occurs .

5 MULTIPROCESSOR SOFTWARE DESIGN

The software produced for the slave processors can be divided into a number of routines corresponding to the sections of theory discussed in the previous chapter . These routines are called by a coordinating routine in a fixed order: each routine ends by broadcasting its results to all the other processors in the simulator . The software for the master processor deals with the communication, both between processors and between the simulator and the host computer . The timing of the simulation is also dealt with by the master processor so that the results are produced in real time .

5.1 MASTER PROCESSOR SOFTWARE

The master processor's main task is to control the data transfer between the slave processors and the transfer of the simulator results to the host computer . The master also deals with the initialisation of the simulator and the addition of errors to the results before their transfer to the host computer . The overall flow of the master processor software is shown in figure 5.1 but each of the four main functions is described below :

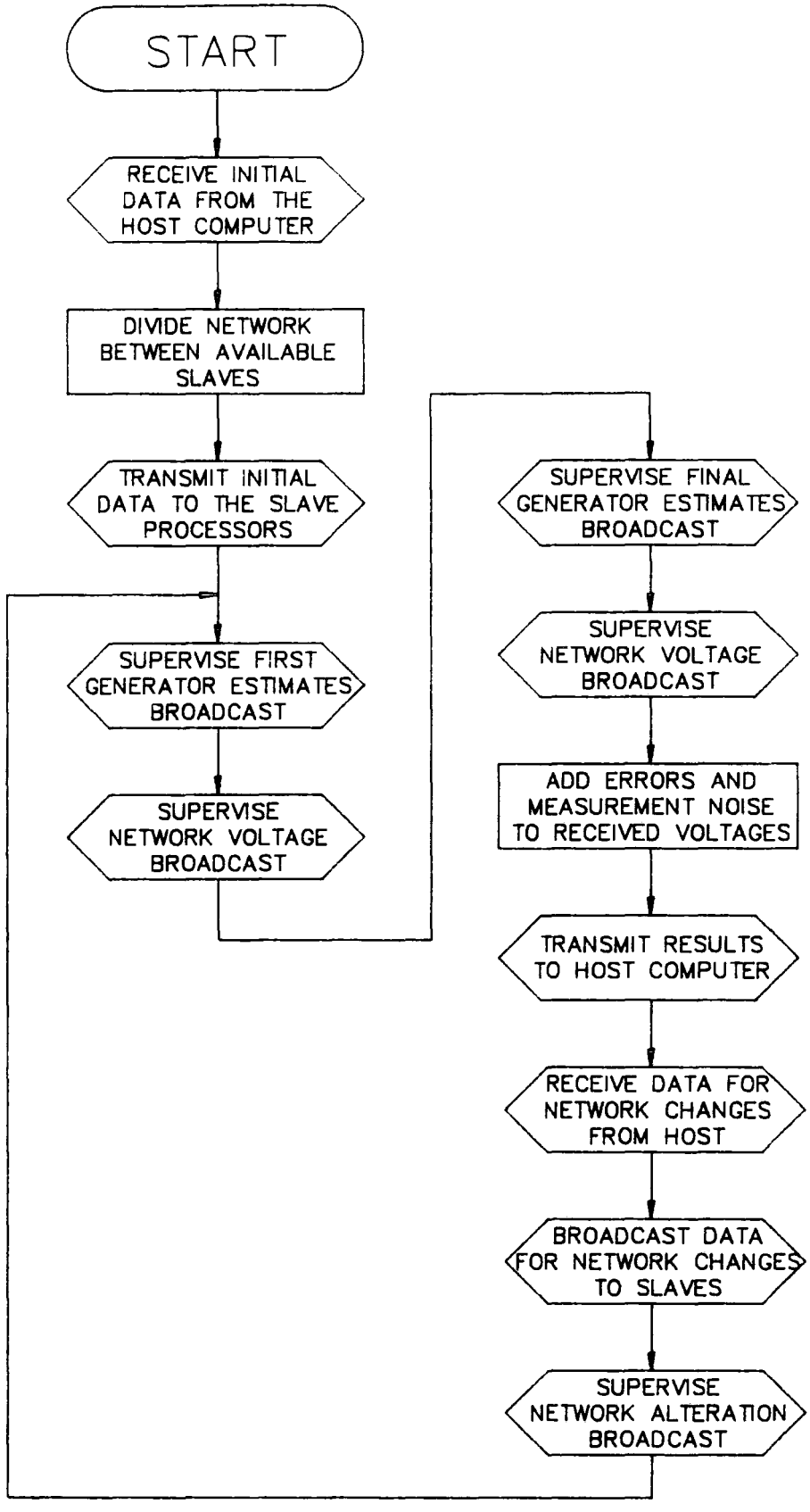


Figure 5.1) Master routine flow chart

5.1.1 SIMULATOR INITIALISATION

The first action taken by the simulator when it is started is to request the network data from the host computer, this data consists of the generator, line and load parameters as given in appendix 3 and is transmitted along the serial or parallel line from the host to the simulator . The host computer also transfers the initial D^{-1} matrix for the system to the simulator . The D matrix is built from the system data and then inverted on the host . However, this is only done once for each system: the inverted matrix can be stored on the host (on disc or tape) for use whenever the simulator is started .

When the system data has been received by the master processor the number of slave processors in the simulator is input . The master processor then calculates the number of buses, generators and loads that each slave is to be allocated . The system is split so that the buses allocated to a processor include those containing the loads and generators allocated to that processor . This reduces the communication required for each time step .

The master processor then resets the communications bus connecting the processors and transfers one area of the split system to each of the slaves . Each slave in turn is sent the numbers of the buses, generators and loads which it has been allocated . The slave is also sent the columns of the D^{-1} matrix which correspond to the buses allocated to it .

5.1.2 BROADCAST CONTROL

When the initialisation of the simulator has been completed the master processor starts to control the use of the communications bus to coordinate the slave processors . Most of the inter-processor communication is in the form of broadcasts; one processor outputs data onto the bus which is received by all the other processors . The rest of the communication is simple data transfer between two of the processors .

All the communication is done under the control of the master processor . Each slave can be put into either 'talk' or 'listen' mode by the master processor . When a slave receives a talk command it outputs its data onto the bus: all the slaves in 'listen' mode receive the data . For a simple data transfer the master processor sets one processor to 'listen' and one to 'talk' and then monitors the transfer: when the transfer is over the master releases both processors .

A broadcast is more complex . Normally all the slaves have data to share with all the other slaves so the master allows each to talk in turn . The master sets all the slaves to 'listen' and then, starting with slave number one, sets each slave in turn to 'talk' . The master monitors each slave's broadcast and, when all the data has been transferred, sets the talking slave to 'listen' and sets the next slave to 'talk' . This process is repeated until all the slaves have broadcast their data .

The broadcast control is also used to keep the simulator running in real time . A timer on the master processor is checked at the end of each time step and, if the simulator is ahead of real time the broadcasts are held up until real time corresponds to the simulator time .

5.1.3 RESULT CORRUPTION AND TRANSFER

The final task for the master processor is the transmission of results to the host computer . However, to simulate the real system accurately, the results sent to the host must have errors similar to those due to the transducers in the real network . If no errors are impressed upon the data the simulator provides no test for any of the state estimation and data validation algorithms needed in any energy management system .

There are two types of error that occur in the real network which have to be modelled by the simulator :

First, the inaccuracy of the measurement due to the limitations of the transducers and the affects of noise on the measurements .

Secondly, the errors caused by transducer failure or failure of components in the data acquisition system .

The first of these errors is simulated by imposing two different multiplying factors upon all the data sent to the host . The first factor is a constant for any measurement and simulates the calibration error of the transducer . The second factor is a random percentage increase or decrease to simulate noise on the measurement . Thus each measurement M is obtained from the corresponding calculated figure C by :

$$M = \frac{(100 + RND(X)) * (Y * C)}{100} \qquad (5.1.1)$$

where Y is the calibration error and X is the maximum percentage noise variation . The RND is a mathematical function which produces an evenly distributed random number but can easily be changed to produce any sort of random distribution required .The second type of error is represented in one of two ways: the measurement can be fixed to any particular value or can be transmitted as a random varying number .

Any measurement can be affected by either of these forms of error by a simple command from the host to the simulator . The calculation of the errors is done in parallel with the network calculation on the slave processors and only the data to be transferred to the host is affected .

The transfer of these measurements from the simulator to the host is via a simple serial or parallel line connecting the two machines . The simulator transfers the data in a fixed order to

correspond to the data scan on the real network and at a rate similar to that achieved by SCADA systems .

Since the calculation required for imposing the errors upon the data is small it can, along with the data transfer to the host, easily be performed in real time . The master processor deals with all the computing necessary while the slave processors are calculating the next set of values .

5.1.4 SIMULATOR CONTROL

The simulator receives control signals from the host computer along the same connection used for the result transfer . The master receives these control commands and acts accordingly . The command from the host takes the form of a single byte which holds the type of change required followed by a series of bytes of data needed to implement the change . For example the command byte 01 corresponds to a generator set point change and is followed by three bytes of data, the generator number, the new power set point and the new frequency set point . The command byte 02 represents a line outage and is followed by a single byte holding the line number .

When the master processor receives a command it takes one of three actions :

For a network change, such as a line outage or change in a constant impedance load, the master processor broadcasts the command to all the slave processors and then controls the broadcast during the matrix alteration .

For a change in the dynamic part of the model, generator set points or load levels, the master processor sends the change to the relevant slave .

For a change in the errors imposed on any measurement the master simply updates the variables in its memory and continues .

The master keeps count of the number of changes and allows only a certain amount per iteration, so the simulator continues to run in real time . If more changes are requested the master postpones them until the next time step .

5.2 SLAVE PROCESSOR SOFTWARE

The slave processors perform most of the actual modelling of the network components . The software written for the slaves can be divided into sections which are called in order by a coordinating routine, each section involving some communication between processors . The communication not only transfers data between the processors but also means that all the slaves are in the same section of the software at

any time . The overall pattern of the slave software is shown in figure 5.2 while some of the sections are shown in more detail in figures 5.3 to 5.5 .

5.2.1 SLAVE INITIALISATION

When the simulator is started the slaves all wait to be put into 'listen' mode by the master processor . When this happens the slave reads in the data put onto the communication bus by the master .

The first items of data received by the slave are the size of the network and the generator, load and bus numbers assigned to it for simulation . From these the slave can calculate the amount of data it is to receive in the form of generator and load parameters and the size of its portion of the D^{-1} matrix . This data is then also read from the bus and the master releases the slave from 'listen' mode and moves on to pass initialisation data to the next slave .

The slave then calculates some of the constants used by the simulator . For example the generator inertia H is inverted so that in the generator calculations a multiplication can be used instead of a division which is far slower . The slave then performs the generator initialisation, calculating equations (4.1.4) to (4.1.8) from the initial set up data .

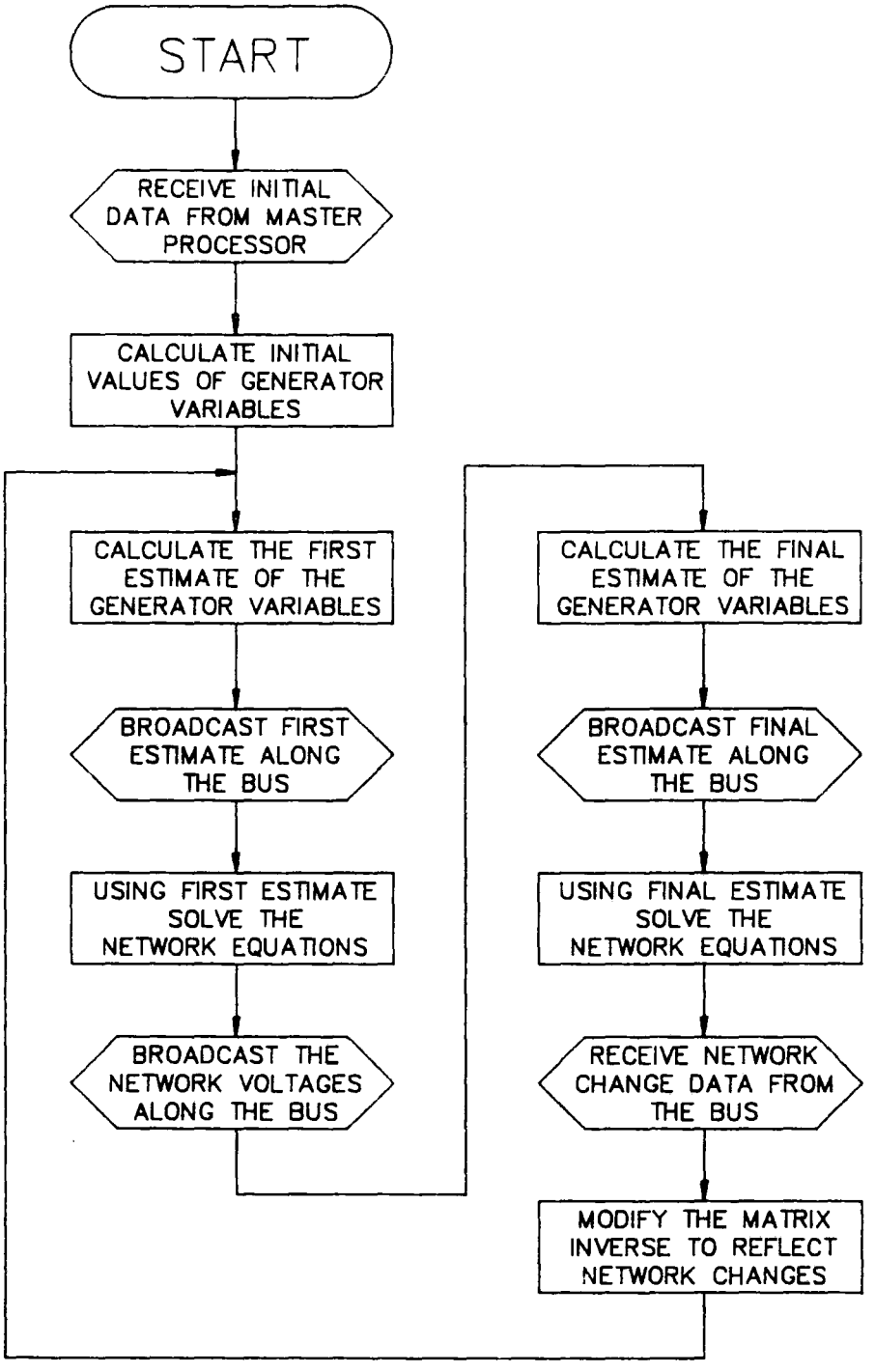


Figure 5.2) Slave routine flow chart

Having performed the initial set up data the simulator continuously loops through the following routines as shown in figure 5.2 .

5.2.2 GENERATOR ROUTINE

The flow of the computation in the generator routine is shown in figure 5.3 . The routine is split into two parts, the initial estimate calculation and the final value calculation . When the routine is entered a flag is tested to determine which of the two parts of the routine is to be performed . The initial estimate is calculated by performing equations (4.1.1) to (4.1.3) and (4.1.9) to (4.1.13) . A new value of E' is then calculated from the constant magnitude calculated in the initialisation and the estimate of the machine angle δ_1 . If the loads in the system are not modelled as constant impedances the value of E' is used to calculate the machine current ITI . This current is then used as an injection by the network routine . The flag is then set to indicate that the next pass through the routine is to calculate the final values .

The slave then broadcasts its calculated values of E' (or ITI) and receives the values of E' (or ITI) for the other generators in the system . The slave performs the broadcast by poling the bus to find out whether it is in 'talk' or 'listen' mode, in 'listen' mode it receives the values broadcast by other

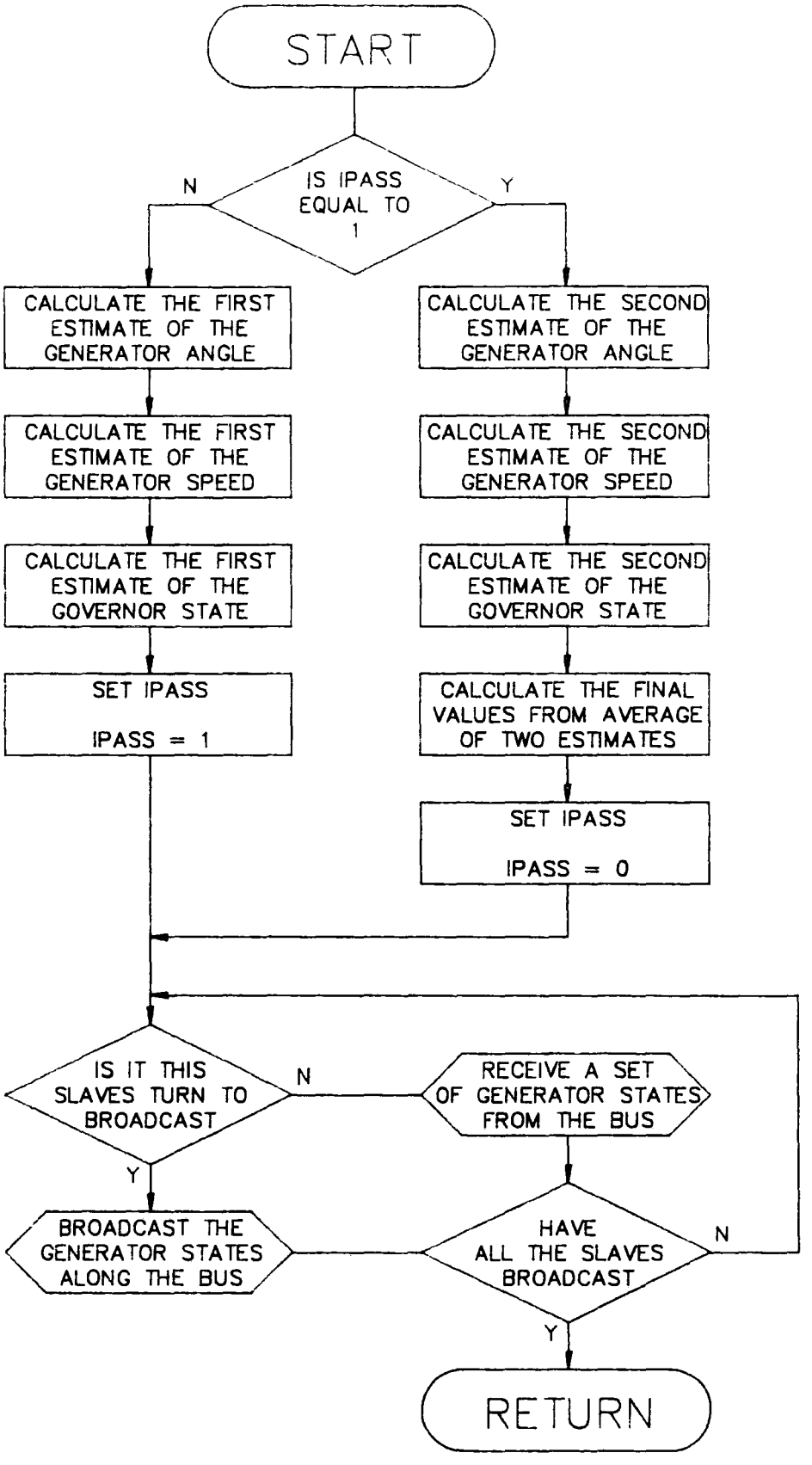


Figure 5.3) Generator routine flow chart

processors while in 'talk' mode it puts its own values onto the bus . The slave then performs a network solution to produce new values for the voltages at the generator buses .

After the network solution is completed the generator routine is re-entered . Since the flag was set at the end of the first pass through the routine calculates the final values . This is done by performing equations (4.1.14) to (4.1.16) . The flag is reset and values for E' are produced and broadcast by the slave .

All the calculation in the generator routine is performed without using any divisions . The divisors in all the equations used are inverted in the initialisation part of the slave software and multiplication is used instead .

5.2.3 LOAD ROUTINE

If the loads in the system are to be modelled separately from the network, not as constant impedance loads, a load routine has to run alongside the generator routine . If a dynamic load such as a predominantly induction motor load is required, the generator algorithm can be tailored to use the modified Euler technique to simulate the dynamic response . However for a more accurate model of the general static loads such as lighting and heating a separate routine is needed .

The general load routine simply has to solve equation (4.2.3) to calculate the load power and then calculate the load current which is used as an injection in the network routine . The load routine is run at the same time as the generator routine, both for the estimate and final value pass through . The calculated load currents are broadcast with the generator currents at the end of each pass so the network routine can use them to calculate the bus voltages .

This routine is also run without using any divisions, the inversion of the nominal bus voltage V_n having been performed in the initialisation . There is no need for a routine to perform the exponentiation (X^n) because the powers used in the routine are 0,1 and 2 . The square is the only power that needs any calculation and can be performed by a simple multiplication .

5.2.4 NETWORK ROUTINE

The network solution routine is run twice for each time step, once during the generator solution and once to produce the network voltages required . The routine involves the solution of equation (4.3.12) if the loads are modelled with the network: if the loads are modelled separately then equation (4.3.13) has to be solved .

The direct network solution is extremely fast in the calculation of the voltages at the nodes in the network . The computation required is small and, in a high level language, the

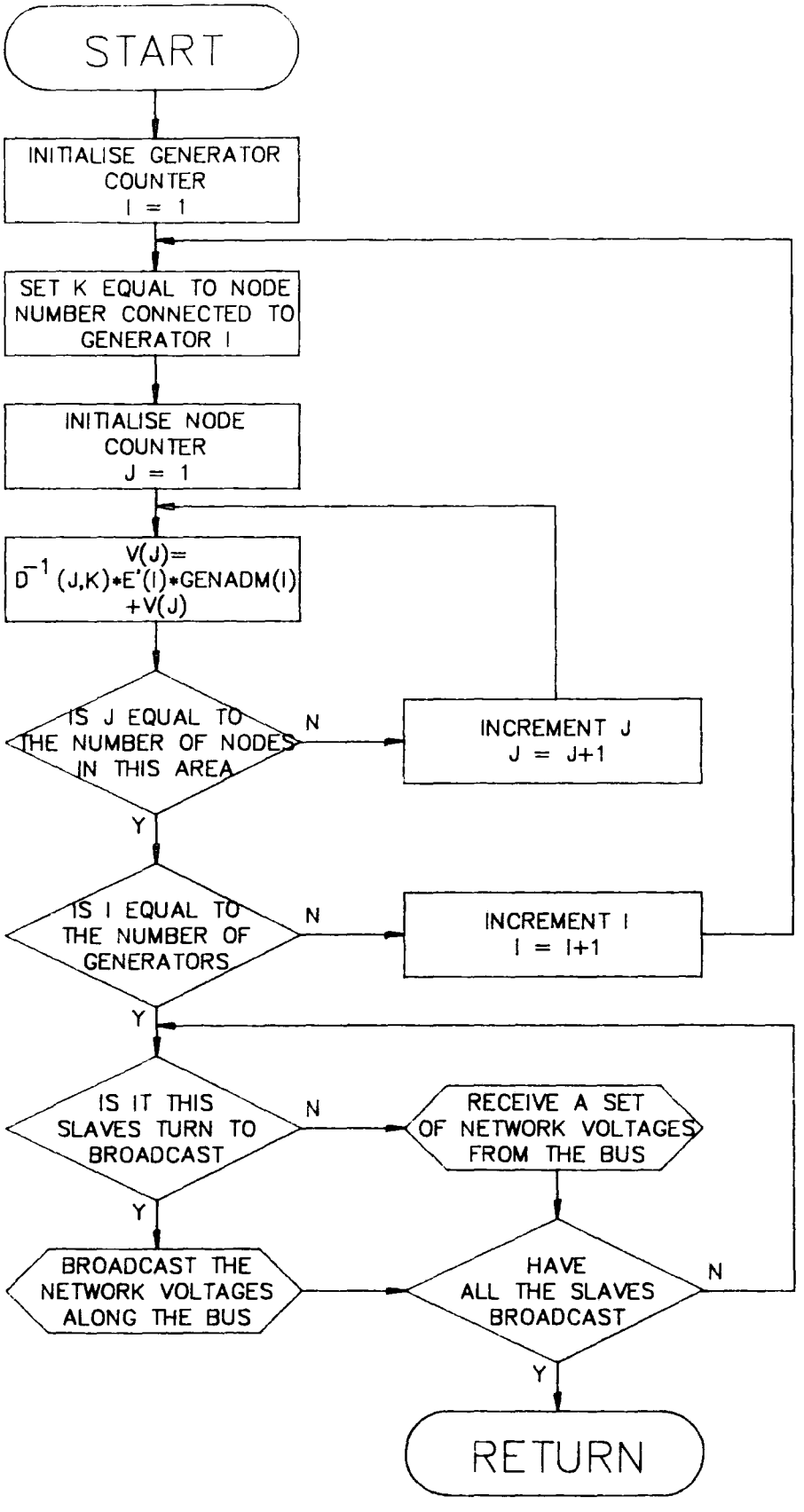


Figure 5.4) Network routine flow chart

size of the program required is also small . For example in FORTRAN the central computation can be written in 3 lines

```
      DO 100  I = 1,NB
      DO 100  J = 1,NC
100   V(I) = V(I) + DINV(I,J)*IINJ(J)
```

where NB is the number of buses, NC is the number of current injections, V is the column vector of nodal voltages, DINV is the D^{-1} matrix and IINJ is the column vector of injected currents .

If the generators are represented as a voltage rather than as an injected current then a further multiplication is required . However, even with this representation the calculation can fit into a small amount of memory which, for certain processors, can increase the speed of computation . When the network routine has finished the processor broadcasts the calculated voltages along the communications bus to the master and the other slaves . The processor then receives the voltages calculated by the other processors as they take their turns to broadcast .

If there is a significant number of buses without current injections in the network the routine can be speeded up both times it is run . The network solution which occurs in the middle of the generator routine only needs to calculate the voltages of the buses with current injections . The other network solution also has to calculate the buses with current injections and must calculate the voltages at any other buses if they are to be transmitted to the host processor for that iteration .

This technique is not available in most of the standard iterative methods for the network solution and it can reduce the solution time significantly . A small amount of extra calculation and memory are used in order to determine which voltages are required at each iteration but this is normally offset by the time saved in not computing the voltages of all the buses in the network .

5.2.5 HOUSEHOLDER ROUTINE

The final section of code in the slave processors deals with the D^{-1} matrix . The network solution requires a copy of the matrix to be available each time it runs . The matrix must be up to date, representing the current state of the network . The Householder routine updates the section of the inverted matrix held by each slave every time a change occurs in the system .

The flow of the Householder routine is shown in figure 5.5 . After each iteration has produced a set of results to be sent to the host, the master processor deals with the changes in the network . Each network change is sent to the slave processors involved, the slaves waiting to receive the type of change and the parameters which follow .

There are two types of change which the slave processor has to undertake, the alteration of generator and load parameters and the alteration of the network topology itself . If the change is a generator or load alteration the slave merely receives the parameters

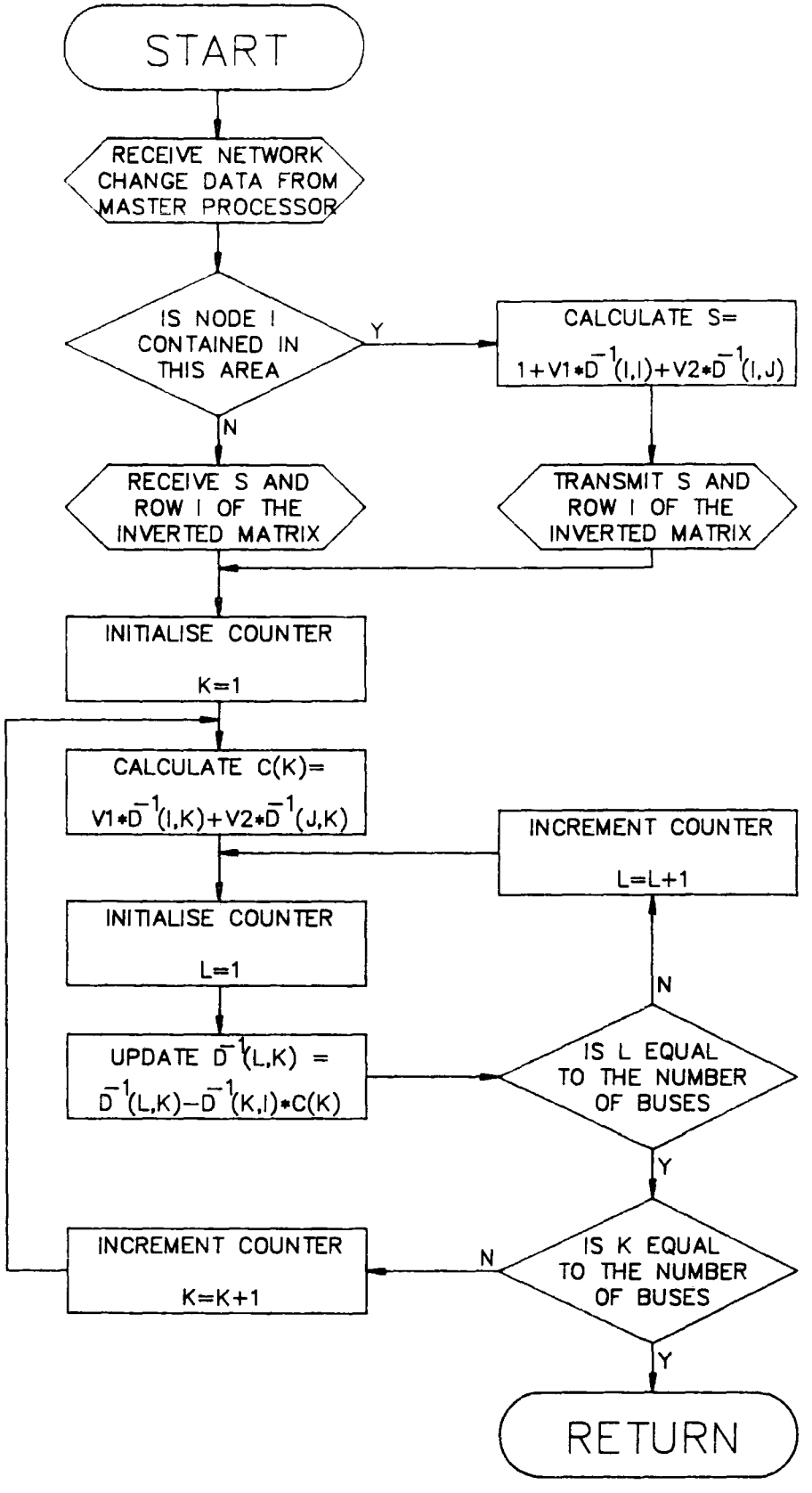


Figure 5.5) Householder routine flow chart

from the host and updates the variables in its own memory . If the change is an alteration to the network the processors perform the householder inverse alteration algorithm .

The matrix alteration is carried out in two stages . The first stage is performed by one slave and then the second is performed by all processors in parallel . First the processor which contains the column of the D^{-1} matrix with the diagonal element representing the node to be altered calculates the value of S as shown in equation (4.4.3) . The inverse of S is then calculated and broadcast, with the column of the D^{-1} matrix of interest, to all the other slaves in the multiprocessor . When all the slaves have all the information the new values in the inverted matrix are calculated by substituting (4.4.3) into (4.4.2) to give :

$$(D + uv^T) = D^{-1} - (D^{-1}u)(v^TD^{-1})/S \qquad (5.2.1)$$

By broadcasting the inverse of S the amount of division performed by the routine is reduced and the execution time is correspondingly reduced . When all the routines have completed the calculation of the altered inverse the master processor starts the slaves on either the next matrix alteration or the next time step of the simulation .

5.3 SOFTWARE FUNCTIONS

Because the floating point hardware can only perform the basic mathematical functions (multiplication, division, addition, subtraction and the conversions between real and integer formats) several of the more complex functions have to be performed in software .

The functions required are the sine and cosine of an angle, and the arctangent and square root of a number . Because some of these functions are used frequently by the simulator the time spent in evaluating them is critical .

5.3.1 SINE AND COSINE

The sine and cosine functions are performed in the generator routine each time new values for E' or I_t are calculated . To obtain the values required as quickly as possible a look-up table is used . The look-up table is simply a list of the values of the function held in memory in the same way as in books of logarithmic tables . When the cosine of a number is required the number itself is used as an index to calculate the position of the table in which the required answer is to be found .

The table used has 4096 elements and holds the cosines of numbers in the range 0 to 360 degrees . The table is calculated on the Perkin-Elmer 3230 in double precision and transferred onto the development machine . For the calculation of the sine of a number the

same table is used and the cosine of the number + 90 degrees calculated .

When the routine is called with a number (x) in degrees of which the cosine is required it first ensures that x lies between 0 and 360 by adding or subtracting 360 as necessary . The routine then performs the following equations :

$$I = INT(x * 11.378) \qquad (5.3.1)$$

$$C_1 = A(I) \qquad (5.3.2)$$

$$C_2 = A(I+1) - C_1 \qquad (5.3.3)$$

$$\cos x = C_1 + C_2 * (x - I/11.378) \qquad (5.3.4)$$

Equation (5.3.1) calculates the index to be used in the table of cosines which is held in the vector A . The two nearest values in the table are then obtained A(I) and A(I+1) and a linear interpolation between these values is used to calculate the final value for the cosine of x (5.3.4) . The value produced by this method is accurate to at least six decimal places and is obtained very quickly . The multiplications and the conversion between real and integer formats are all performed by the floating point unit: the division is performed by using the reciprocal of the constant and multiplying .

5.3.2 ARCTANGENT

The generator routine also requires the calculation of an arctangent during the initialisation procedure . However, since this is only performed once and does not affect the timing of the iterations, once the simulator is running the function can be evaluated using a numerical approximation such as a Taylors series expansion.

5.3.3 SQUARE ROOT

The calculation of the square root of a number is required in both the generator and load routines to calculate the magnitude of a complex quantity . Because it is so widely used the execution time of this routine has a significant effect on the speed of execution of the simulator . Unlike the sine and cosine routines there is no finite range of values which have to be catered for, although the likely range of numbers is between 0 and 2 because of the use of the per unit system .

Due to the way floating point numbers are represented by the computer the square root problem can be split into two sections . The floating point representation used is to hold the numbers as a power of two multiplied by a fraction between ½ and 1 :

$$x = 2^n * f \qquad (5.3.5)$$

The square root is then calculated by :

$$x^{1/2} = 2^{n*1/2} * f^{1/2} \qquad (5.3.6)$$

The routine takes the real number and separates the exponent (n) from the mantissa (f) . If n is an odd number then n is incremented by 1 and f is halved: n is then halved and stored in the exponent of the answer while the square root of f is calculated and stored in the mantissa of the answer . Since the values of f lie in a fixed range the square root of f can be calculated by using a look-up table in exactly the same way as the sine and cosine values were calculated .

The possible values of f range from ¼ to 1 and a table of 4096 values was calculated on the Perkin-Elmer and transferred onto the development machine . The accuracy of the routine depends on the size of the number whose square root is required . However for the range of values 0 to 2 the routine still produces answers accurate to six decimal places . Also the answers produced by the routine are always normalised, that is the mantissa always lies between ½ and 1, which is the most accurate way to store the numbers and helps in the use of the floating point hardware .

5.4 HOST SOFTWARE

The software required on the host computer splits into two parts . One part deals with the initialisation of the simulator and the calculation

of the initial D^{-1} matrix: the other part has to deal with receiving measurements from the simulator and passing commands from the control software or operators to the simulator .

The initialisation software has to read the network data either from a file or interactively from the operator . If the network has previously been simulated the D^{-1} matrix can be read from the disc . If not, then the network admittance matrix (Y) must be built up from the network data and then the D^{-1} calculated from it using matrix inversion routines . The host must then pass the network data and the D^{-1} matrix to the master processor in the simulator via the parallel or serial link between them . If the matrix inverse has to be calculated it should be stored on disc or tape to save time if the network is simulated again .

Having initialised the simulator, the host computer then has to receive the measurements from the simulator and pass these on to either the control software or the operator, depending on how the simulator is being used . The flow of control information from the host to the simulator also has to be dealt with by the host, any instructions from the operator or control software having to be translated into the correct format and transmitted to the simulator .

The transmitting and receiving of data across the link between the host and the simulator can normally be performed from a high level language using standard input/output commands if the link is connected to one of the host's standard ports . At present no checking is done on the received data . Noise is not likely to be a serious problem to the data transfers,(it is more likely to occur in the real network than in the

simulator) and the data validation routines in the E.M.S. should filter out any erroneous values that occur .

6 MULTIPROCESSOR HARDWARE DESIGN

The hardware required for the simulator was designed in two stages . First the processor on which the simulator was to be based was chosen . Then the software was developed and the design completed to allow the software to run as efficiently as possible . By using this approach, the software could be developed in a form suitable for the processor chosen without a fixed multiprocessor design having any detrimental effect on the software design .

Several previous researchers have looked at the parallel solution of power system problems, I. Durham et al. [43 & 44] and C. Pottle [45] . But much of the work was done on existing hardware used for general purpose parallel processing and not necessarily optimally configured for power system problems .

6.1 MULTIPROCESSOR COMPONENTS

The choice of microprocessor from among the wide variety on the market was limited by the requirements of the simulator itself . The floating point representation needed for simulation had to use at least 32 bits . Therefore to reduce the time spent in moving the data around, the processor had to be able to cope with 32 bit data structures as easily as possible . The number of processors which have 32 bit data words and can deal with 32 bit words with a single instruction is still small and, of

these, two were chosen for closer study . These two were the Motorolla M68020 and the Inmos Transputer .

Another method of developing multiprocessor algorithms was also looked at, using a standard minicomputer and simulating the performance of a multiprocessor, and this method was used for the development of some of the software in the early stages of the project .

6.1.1 MULTIPROCESSOR EMULATION

A system of emulating the multiprocessor's performance was developed on the Perkin-Elmer minicomputer to give an indication of the suitability of different algorithms for solution by multiprocessors . The system worked by having a central coordinating task which shared the central processor unit (C.P.U.) time between the tasks which would normally be run on separate processors . This worked in much the same way as the part of the operating system which shares the C.P.U. time between the computer users, giving all users the same facilities as if they each had a smaller computer dedicated to running their own problems .

The tasks were written in FORTRAN in exactly the same way as if they were to be run on separate processors and had an interrupt routine added . The central task could then start each task at any time and the task would interrupt itself after a preset amount of time and pass control back to the central task . The central task would start all the tasks in turn, dividing the time equally between

them . When the final task had returned control to the coordinator the process started again with the first task unless all the tasks had finished .

Communication between the tasks was done by using an area of memory shared by all the tasks, which any of the tasks could write to or read from . The area of memory was set up as a global common block so that to read from or write to it the task merely had to use the variables defined in the common block . The accesses to the global common block could easily be counted so as to provide an idea of the amount of inter-processor communication required . This method of modelling the effect of a multiprocessor on an algorithm had several factors in its favour ;

There was no need for specialist hardware, so the algorithms could be developed and evaluated without the delay involved in the design and production of a multiprocessor system .

The use of the technique allowed the algorithm to be tested before the hardware was designed: the hardware could then be designed to perform as well as possible with the algorithm .

Because of the simplicity of the method and the use of standard hardware any errors occurring in the algorithms could

be attributed to the software rather than the possibility of hardware errors .

There were also some disadvantages with the system . In order to emulate properly the affect of a multiprocessor on the algorithm the time allocated to each task for each cycle of the coordinator had to be very short . The tasks also had to return control to the coordinator if they output any data to the global common block . This was necessary so that the other tasks had a chance to read the data before it was changed . All these control changes took time, so the system was quite slow . The second problem was that in order to get the tasks to run correctly they had to be run at a higher priority than the time sharing system on the minicomputer so that they were not interfered with . This meant that while the system was running no other users were allocated any C.P.U. time and for them the computer appeared to stop .

This method was used to help develop, and in some cases discard, the methods discussed in Chapter 4 . However, having developed the direct method the hardware was built so that the algorithm could be fully tested and developed on a true multiprocessor system . Although the decision on which processor to use had to be taken towards the start of the project, the development of this system of emulating the effect of using multiple processors was still useful . It allowed the software to be developed in a form that could readily be transferred onto the final hardware . At the same time the overall structure and the design of the hardware could

be left until the detail of the software algorithm had been developed and tested .

6.1.2 TRANSPUTERS

One of the two processors looked at closely was the Inmos Transputer . The Transputer is not strictly just a microprocessor . It is a single chip containing a processor, memory and interfaces to peripherals and other Transputers as shown in figure 6.1 . It has been designed specifically for multiprocessing tasks and can use the language OCCAM which supports the concepts of concurrent processes communicating via interprocess links .

The Transputer is an extremely powerful processing element for building parallel processing systems and has several points to its advantage ;

The architecture of the Transputer is designed for direct connection to other Transputers for parallel systems . The four serial links can be used to connect one Transputer to up to four others and , even though the links are serial, allow high speed communication .

The processor has a 32 bit data structure and operates at high speed (up to ten million instructions per second) and is designed specifically to support concurrent tasks .

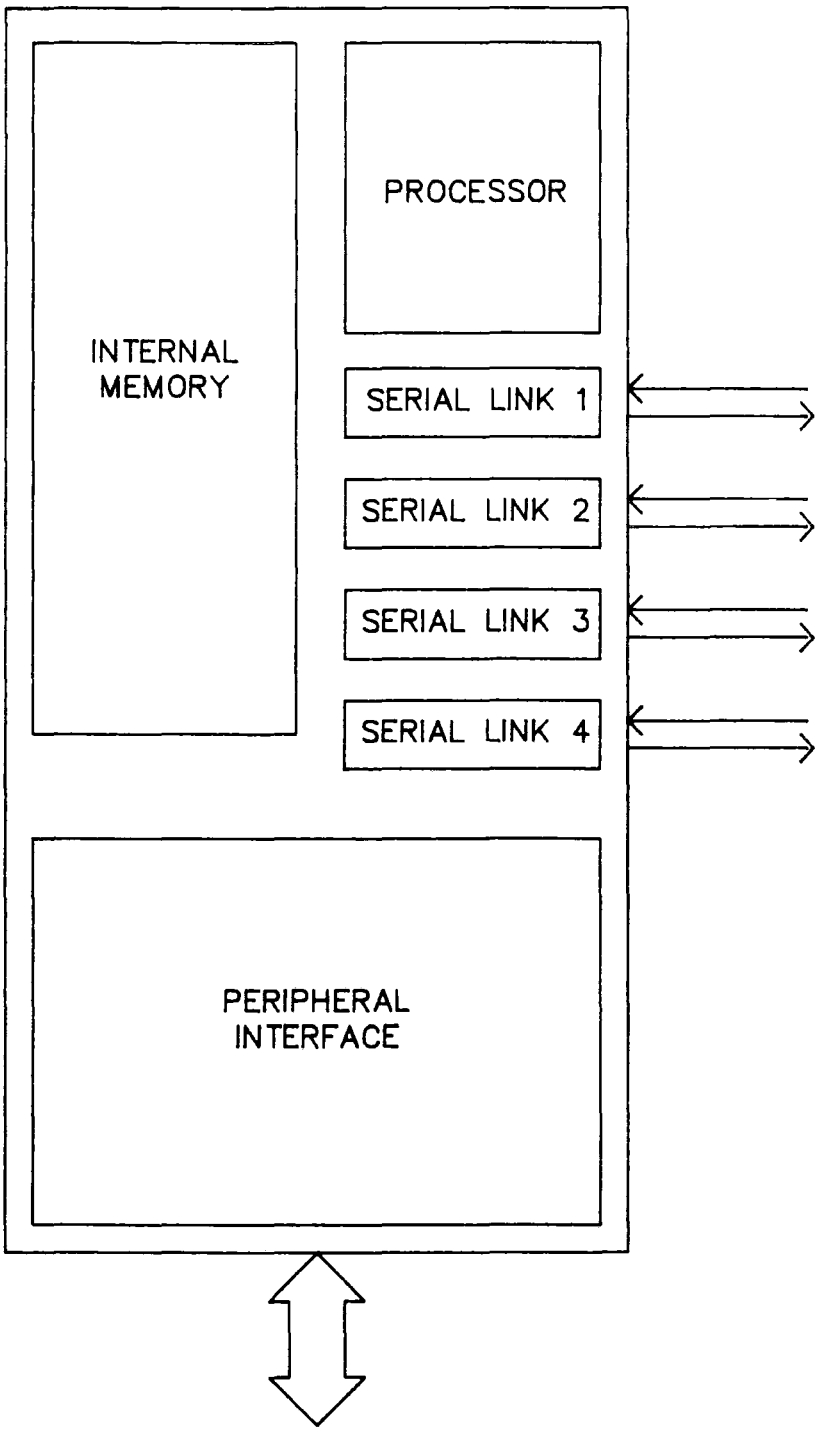


Figure 6.1) Transputer architecture

The language OCCAM is a high level language which is closely coupled with the Transputer . OCCAM produces efficient code both in terms of size and speed and facilitates the programming of parallel tasks .

Each Transputer can support more than a single task, so that a parallel algorithm can be run as a number of tasks on a single Transputer or as a single task on a number of Transputers . Both configurations run in exactly the same way and produce the same results .

These advantages make the Transputer a very useful building block for any form of multiprocessor system . However, for use in the power system simulator there were two major problems with the Transputer :

When the initial study was done the Transputer was still being designed and the launch date was uncertain . Emulators were available to enable software to be developed but the cost and availability of the actual Transputers were uncertain .

No information about future development of the floating point capability of the Transputer was available . The amount of calculation required for the simulator suggested that some form of external floating point hardware would be required for either the Transputer or the Motorolla processor . Because the

Transputer was not available the ease of using existing floating point hardware could not be ascertained .

It is likely that, given time, both these problems will be rectified, the Transputer should soon become widely available and further developments in the field are being planned . When these events occur the Transputer will certainly become an extremely useful component in the design of both standard computers and multiprocessors .

6.1.3 M68000 AND M68020 MICROPROCESSORS

The second processor to be considered was the Motorolla M68000 family and especially the M68020 . The M68000 family is a series of microprocessors all of which have a 32 bit internal data structure . Although the internal data size was 32 bits the size of data that could be handled externally to the processor varied . The M68000 could only read or write 8 bits while the M68010 could handle 16 and the M68020 32 bits .

The M68000 family, although it is not specifically designed for use in multiprocessors, had several points in it favour at the time of the study ;

The M68000 was available immediately and, with the use of a UNIX development system, could be programmed in a number of high level languages including FORTRAN and C . The M68000 was very widely used and the tools, both in hardware and software, to aid development of M68000 systems were extensive .

Any code written for the M68000 would work on the M68020 whose release onto the market was imminent . The only difference in the execution of the code would be the faster execution time on the M68020 . The M68020 provided some additional facilities which could speed up the execution time still further .

There was floating point hardware being designed specifically for the M68020 by Motorolla which would be available before the end of the research project . Other manufacturers were also developing floating point hardware for the M68000 family so there would be a choice of components when the system was designed .

The University had a M68000 development system and produced its own processing board using the processor for use within the University . Equipment was also available for the design and production of printed circuit boards for any hardware that had to be built .

The only disadvantage with the M68020 was that the method of connection of the processors into a system would have to developed to enable them to communicate at speeds high enough for the simulation to be feasible .

The decision finally taken was to begin development on the Perkin-Elmer, using the software to model the affects of a multiprocessor, but also to use the M68000 development system to produce code for the final simulator . The final simulator hardware would be based upon the Motorolla M68020 using hardware floating point and developing some form of communication between the processors .

6.2 MULTIPROCESSOR CONFIGURATIONS

There are several methods of configuring the processors in a parallel processing system . Each of the configurations has its advantages and disadvantages, depending on the type of problem being solved on the multiprocessor . Four configurations were looked at, three basic forms and one more complex . The complex form was the CM* multiprocessor developed at Carnegie-Mellon University in America on which much of the multiprocessor power system work had been done . The simpler forms were the linear, radial and grid configurations .

It is not only the hardware that has a configuration: any parallel algorithm will also a particular form . The configuration is based upon how the separate parts (whether hardware or software) are arranged and

how they communicate with one another . In general the form of the algorithm and the form of the processor structure should match: if the algorithm is radial in structure then the best results will be obtained on a radial multiprocessor . If they do not match, the system will not be as efficient as possible because the algorithm will have to be modified to fit onto the processors .

6.2.1 CM* MULTIPROCESSOR STRUCTURE

The CM* multiprocessor consists of a series of processors connected in a tree like structure . The processors are divided into a number of clusters, each cluster containing up to 14 processors . The processors in a cluster are connected by a communications bus which allows each processor to access the memory belonging to another processor . The clusters are then connected in exactly the same way, one cluster being able to access memory in another cluster via the inter cluster bus . The controllers on the buses mean that a processor simply issues a read command from memory and, whether that memory belongs to the processor, another processor in the cluster or a processor in a separate cluster, the controllers provide the data . The time taken for a read or write to memory is longer if the data has to travel from another processor or cluster .

The advantage of this type of connection is that information which is accessed regularly is kept within the memory of the processor . Less frequently used data is kept within the cluster and

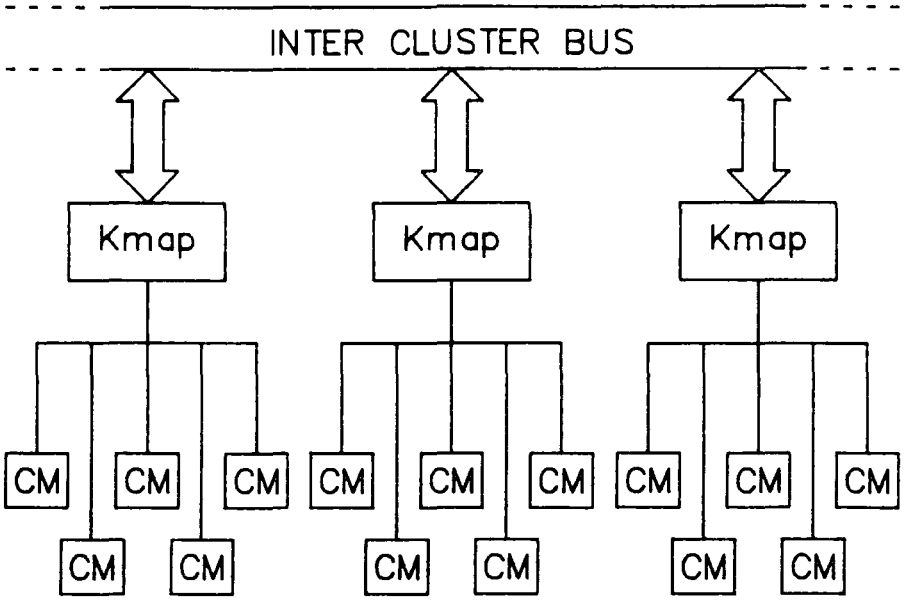


Figure 6.2) Structure of a CM * multiprocessor

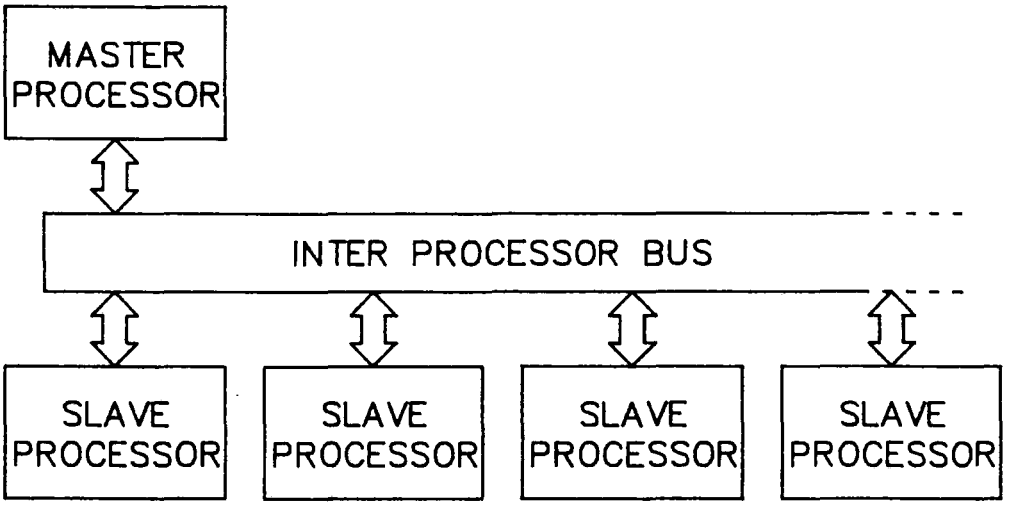


Figure 6.3) Linear multiprocessor configuration

data which is only accessed occasionally is kept in other clusters . This means the the access time is, on average, quite small . Also the system is expandable , more clusters can be added whenever they are required .

The disadvantage is that if the problem requires data to be transferred equally between all the processors some of the transfers take much longer than others because of the distance between the processors in the tree structure .

The tree structure for multiprocessors can be expanded further than the two levels used by the CM* . Three or more levels can be used but the more levels that are used the further some of the base level processors are from each other . This means that the amount of hardware that is used in some of the data transfers increases and the transfers between the processors in the base level clusters take more time .

6.2.2 LINEAR MULTIPROCESSOR STRUCTURE

Each of the clusters of the CM* is connected in a simple linear arrangement . This form of connection requires just one connecting bus with all processors attached to it, as shown in figure 6.3 . Often one processor (the master) has control over the bus while the rest (the slaves) have to wait for instructions from the master to allow them to use the bus . This system of connection has the

advantage that any processor can communicate directly with any other processor in the system .

The disadvantages with a linear connection of processors are that only one set of data can be transferred at any one time and that if the communication bus breaks down there is no alternative data path . Also there is often a restriction on the number of processors that can be connected to a single bus, and therefore a restriction on the size of the multiprocessor and the problems that can be solved using it .

Some bus structures allow the broadcast of data, one processor being able to pass data to all the other processors in the system simultaneously . If this is possible and the algorithm requires large amounts of data to be shared between processors the linear configuration can provide a simple yet effective form of multiprocessor arrangement . More than one bus can also be connected in parallel to all the processors to provide an alternative data path if required .

6.2.3 RADIAL MULTIPROCESSOR STRUCTURE

The second of the basic types of multiprocessor structure is the radial arrangement as shown in figure 6.4 . A central or master processor is connected via a number of buses to slave processors . The master processor has control over all the buses and can request data from the slaves . This allows all the slaves to communicate

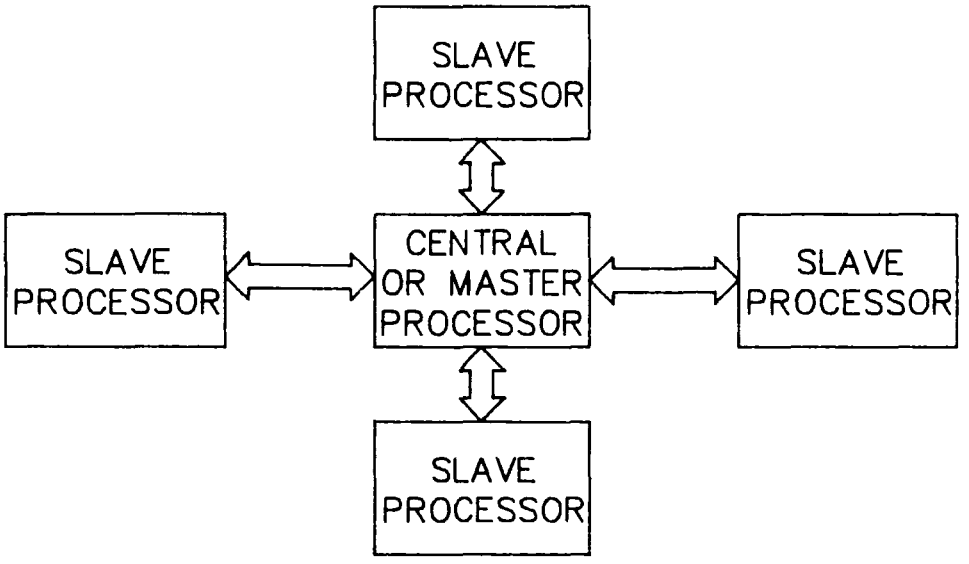


Figure 6.4) Radial multiprocessor configuration

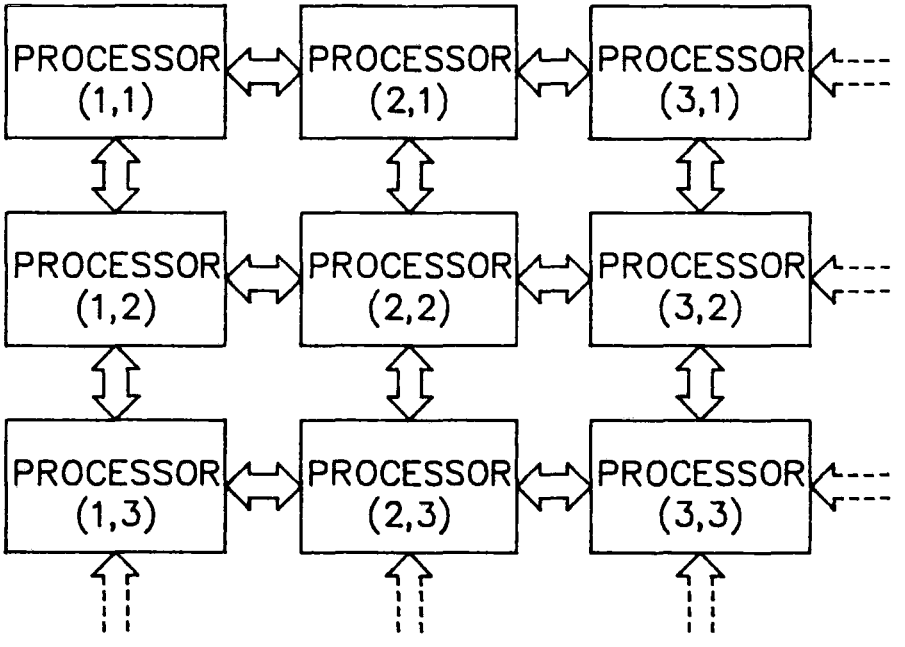


Figure 6.5) Multiprocessor grid configuration

directly with the master in a single transfer and communication between two slaves only requires two transfers . Depending on the form of the bus structure the system can allow more than one transfer at a time, all slaves sending or receiving data at any time, and the master can broadcast to all of the slaves .

The number of slaves connected to the master is limited but the multiprocessor can be built in a series of levels . Just as the CM* has two levels of linear connections, a set of master processors could be connected via buses to a higher level master . If a bus or slave processor fails in a radial network the master can redistribute the tasks between the remaining slaves and the system keeps working at reduced efficiency . If the master processor fails nothing can be done .

This sort of configuration is ideal for some problems, for example the diakoptic solution discussed in chapter 3 is radial in structure . A solution for part of the system is calculated by each of the slave processors, the master then takes these solutions and combines them to produce a solution for the entire network .

6.2.4 GRID MULTIPROCESSOR STRUCTURE

The final configuration looked at was the multiprocessor grid as shown in figure 6.5 . In this setup each processor is connected to its four neighbours and can communicate with them directly . There is no master processor controlling the buses and any processor can

communicate with any other processor . To do this however the processors have to be able to receive data from one neighbour, look to see which processor in the network it is meant for, and then pass it on towards the correct processor . In a large grid the transfer of data between processors may have to travel through many other processors on its way .

The distance between processors can be reduced by several methods . The processors at the top and bottom of the grid can be connected to form a cylindrical array and then the end processors joined to form a torus . Alternatively the processors can have six connections each and form a three dimensional box type grid . Having six or more connections to each processor can cause problems in the physical connection of the array .

The Newton-Raphson method discussed in chapter 3 possesses a grid structure suitable for use with a two dimensional grid multiprocessor . The network is split into parts which then communicate with their neighbours to iterate until they produce a solution for the whole network .

6.3 PROCESSOR COMMUNICATION

The other element to be decided in the design of a multiprocessor is how the communication between is to be achieved . There are really two options for the method of communication , either to use a system of shared

memory areas or just to have a communications bus connecting the various processors .

6.3.1 COMMON MEMORY SYSTEMS

A common memory system will have some or all of the processors possessing an area of memory which is accessible to the other processors in the system . The processors are still connected by some form of bus but the processors just have to issue a normal read or write command to access the shared memory belonging to another processor .

The advantage with memory connected systems is that the communication between processors does not have to be synchronised . The processor with the shared memory can continue performing its tasks while another processor is accessing its memory . However problems can arise with memory contention - when more than one processor is trying to access a particular area of shared memory - and the hardware to cope with contention can be quite complex and slow the system down .

6.3.2 BUS CONNECTED SYSTEMS

In a bus connected system the bus between the processors can be used in two ways, either as a synchronous link or as a channel for

direct memory access between processors . Some buses require a master processor to be constantly in charge of all communication taking place .

For a synchronous link this means that for communication to take place the sending and receiving processors both have to be ready to undertake the transfer . In some cases where the master processor is neither the sender nor the receiver it oversees the transfer, therefore there must be three processors actively involved in the transfer . For some applications the synchronisation between processors obtained using this method is useful in keeping the separate tasks in step .

Using the bus as a DMA link between the processors means that one processor can halt the execution of another processor and forcibly read or write to the halted processor's memory . This type of transfer is useful for transferring large amounts of data between two processors but is similar to the shared memory system and has the same sort of problems .

6.4 DEVELOPMENT HARDWARE

Once the software had been developed on the emulator on the Perkin-Elmer the design of the multiprocessor could be finalised . The structure of the direct algorithm was such that most of the data transfers between the portions of the algorithm were broadcasts to all processors rather than simple processor to processor transfers . The linear multiprocessor

structure is the one which best deals with these data broadcasts because each processor is directly connected to all the other processors in the system .

Because of these broadcasts and because the quantities of data transferred were small it was decided to connect the linear system with a communication bus rather than using shared memory . This configuration seemed to fit best the type of algorithm that had been developed on the Perkin-Elmer .

Two separate multiprocessor systems were used to try the algorithm on: an existing M68000 processor board design could be connected directly into a linear multiprocessor while the M68020 board and its extra hardware were being developed and built .

All the development for both the multiprocessors was done on a UNIX development system which allowed the algorithm to be written in C, FORTRAN or ASSEMBLER or a mixture of all three . The resulting code could be compiled and tested to some extent on the development system before being downloaded onto the multiprocessor for final testing .

6.4.1 M68000 SYSTEM WITH IEEE BUS

Within the University there existed a M68000 board design that was used widely for various high speed computing requirements such as control and data acquisition . A number of these boards contained

enough hardware to run the developed algorithm and be connected in a linear fashion as required .

The processor board itself contained a M68000 microprocessor, a quarter of a megabyte of RAM, two serial lines and in IEEE bus . There was also a small amount of ROM containing a monitor program which allowed the board to communicate, via the serial lines, with the UNIX development system and a terminal . The monitor also contained debugging facilities such as tracing the execution of code a single instruction at a time and setting breakpoints to halt the execution at specific points . The quarter megabyte of memory was enough for the code (which only occupied a few kilobytes) and enough data for quite large networks to be tested .

The IEEE bus is a standard communications bus: it allows the connection of up to 16 digital devices of any sort which possess the correct interface . The bus can transfer eight bits of data at a time, so it takes four transfers to communicate a 32 bit word between two processors . The bus allows any one device on the bus to transmit data to any or all of the other devices simultaneously . One of the devices, the master, has to have overall control of the bus and has to oversee all the communication .

The master processor can, at any time, take control of the bus and set the other processors to one of three modes; 'listen', 'talk' and 'passive' modes . If a processor is set to listen mode it can receive data from the bus . In talk mode the processor can put data onto the bus . Passive mode means that the processor can neither

receive nor transmit using the bus . When the master processor has set all the slave processors to the required state it releases the bus so that the communication can take place .

Only one processor can be set to talk mode at a time and this processor places its information on the bus . The handshaking (that is the protocol which coordinates the use of the communications bus) on the bus ensures that all processors in listen mode have received the information before the next piece of data can be placed on the bus . This method of communication keeps all the processors synchronised because, unlike a shared memory system, all processors involved in the communication must perform either read or write instructions before the system can continue .

Using this system the direct algorithm was programmed in assembler, using software floating point routines . The speed of execution of the algorithms was not as fast as required because of the length of time spent performing mathematical functions . However the algorithms were optimised as far as possible while the M68020 system was designed and built .

6.4.2 M68020 SYSTEM WITH FLOATING POINT HARDWARE

The M68020 processor board was simply an upgrade of the M68000 board used initially . Room was left on the board for the maths co-processor developed by Motorolla and more memory was fitted . It was decided not to use the IEEE bus because of the speed restrictions and

the limit of eight bit transfers at any time . The system at this stage needed two further components to develop it into a system suitable for the task of power system simulation, a hardware floating point unit and some form of inter-processor communication bus .

For the hardware floating point unit it was decided to use the Weitek WTL 1164 and 1165 chips . These gave far faster calculation than the Motorolla co-processor although they did not perform the more complex mathematical functions . The architecture of the Weitek chips is shown in figure 6.6: there are two chips to perform the mathematical operations, a multiplier and an arithmetic logic unit . Both chips have the same structure, having two input registers, A and B, and an output register . The chips are controlled by a number of inputs :

Six input lines are required to define what type of function is required of the floating point chip, for instance whether a multiplication or a division is needed .

Four inputs are used to determine the load type . Whether the A or B register is to be loaded and whether the data to be loaded comes from the data bus or from the result register of one of the chips .

One input is used to signal the chip that the result is to be unloaded onto the data bus .

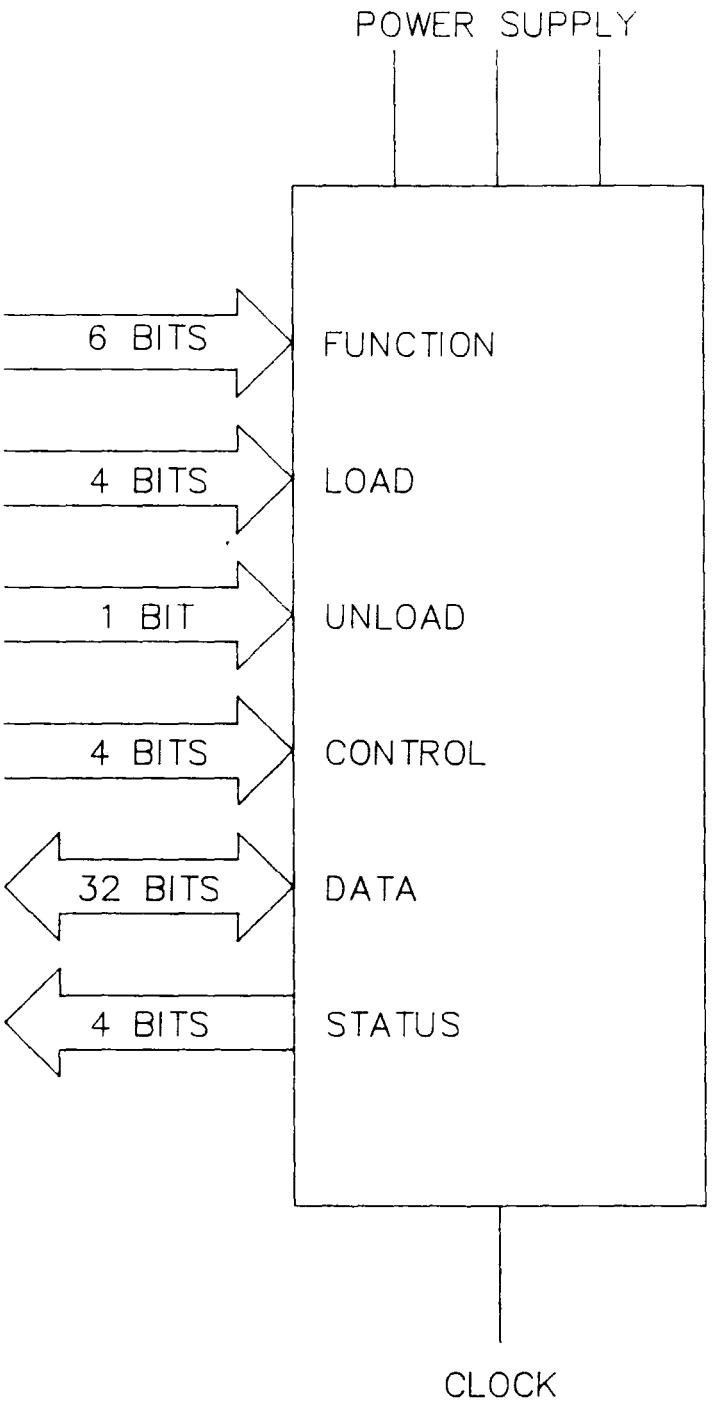


Figure 6.6) Weitek floating point architecture

Four inputs are used to control the sequence of events that make up each of the floating point operations . These inputs have to be changed several times during each mathematical operation so some hardware is needed to provide the correct inputs at the correct time .

The chips are also connected to the power supply, clock, and the 32 bit data bus . Along with the data output onto the data bus the chips also output 4 status lines which report underflows, overflows and errors in operations .

The board designed for the floating point chips is shown in figure 6.7 . All thirty two bits of the data bus are connected, via buffers and latches, to both chips so that data can be loaded and unloaded directly onto the data bus . Only part of the address is used . The board is mapped into a megabyte of memory space, which means that when the processor reads or writes to any part of that megabyte of address space it accesses the floating point board rather than memory . Accordingly only twenty of the address bits will ever change when the board is accessed .

Of these twenty bits, only eighteen are used and these are split into two parts . Eleven bits go directly to the chips and are attached to the function,load and unload lines . The other seven bits are used to set up a 12 bit counter . This counter is driven by the on board clock and counts up from the initial value set from the address lines . The output of the counter is itself used to address a microcode read only memory (ROM) which controls the remaining

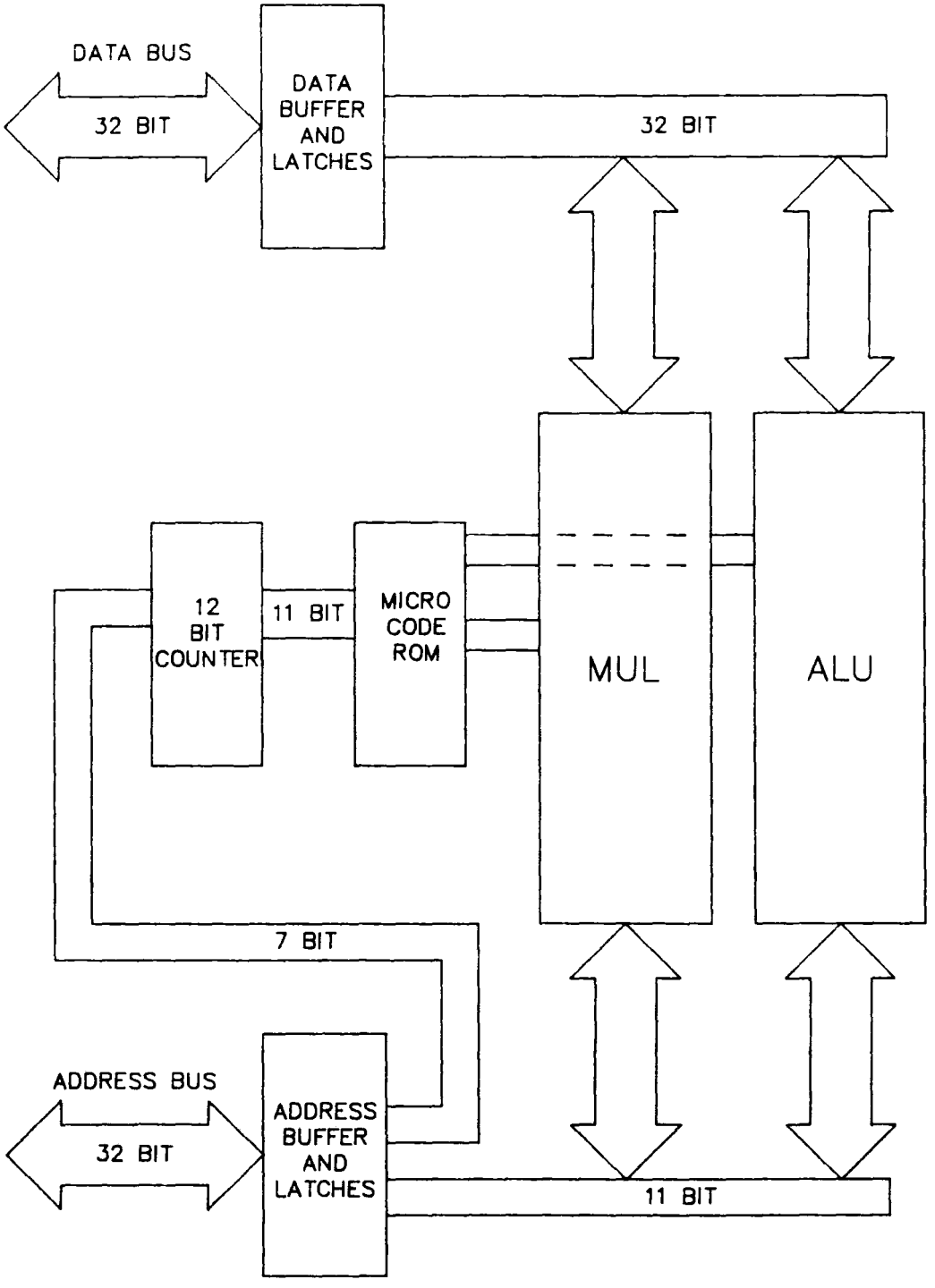


Figure 6.7) Block diagram of floating point board

inputs to the chip . The ROM is programmed with the sequence of control inputs needed to perform the various operations . For example, the multiply function requires different control signals from the add function, so the code for each function is set up in different parts of the ROM: the 7 bits of the address bus, which set up the counter, determine which function is performed .

The microcode also controls the latches on the data bus . This is necessary because the time that the result is available from the chips is very small, so the time when it is latched onto the bus has to be very precise . It is possible to program more complex functions into the microcode ROM . Provision was made for the board to have its own area of memory so that the floating point board could perform complex functions such as polynomials while the microprocessor was performing other tasks . The generator routine could probably be calculated almost entirely on the floating point board, but the time to develop the necessary microcode would be great and the time saved would be small . Tests performed while the simulator was running showed that the floating point processor was active for up to eighty percent of the time during the generator and network routines .

To drive the floating point board the processor merely had to read or write the data to certain points in the megabyte of memory area taken up by the board, the address accessed by the operation determining the function performed . This could be performed from the high level language C using some of the macro functions available, so

the simulator could be written in a high level language rather than in assembler .

The problem of the communication between processors was solved by using a bus similar in operation to the IEEE bus . The bus had the advantages that it could operate at 4 megabytes per second and transferred 32 bits at a time instead of the 8 bits used by the IEEE bus .

This final set of hardware using the M68020, floating point board and high speed communications bus was the hardware on which the final version of the simulator was written and fine tuned to achieve as high a speed as possible . The only problem found with the hardware was the heat dissipated by each of the floating point chips when they were run at high speed . This meant that the air flow through the rack in which the boards were mounted had to be increased by using two fans . Both the microprocessor and the floating point units were run at their rated speeds but a speed up could be achieved by driving the M68020 at a higher speed . If this is done the heat dissipation increases but for certain applications it might prove worthwhile .

7 RESULTS

Most of the development of the algorithms was done on a standard Perkin-Elmer 3220 minicomputer but, because the algorithms had to be proved on a truly parallel machine, the final algorithms were then transferred onto the M68000 system made up of one master and two slave processors . In transferring the algorithms from the Perkin-Elmer they were also translated from 'FORTRAN' into 'C' because of the ease of producing code directly usable on the processor boards using 'C' . With the development of the M68020 system the algorithms were rewritten to use the floating point hardware so that the results for all three systems could be compared .

These comparisons gave close numerical answers, but not as close as expected . However, during the testing of the floating point board it was found that the commercial 'C' floating point software used by both the UNIX development system and the M68000 multi-processor system contained an error . This error could affect the results slightly so a set of routines was written to perform the floating point functions affected . An exact match between these comparisons could not be expected because all three machines used different formats for storing the floating point numbers which altered the accuracy possible from the floating point arithmetic . The comparisons were run with both the IEEE 5 and 30 node test networks and the results obtained were all within the range possible due to the format differences .

Having established that both the M68020 hardware and the parallel algorithm worked, a series of tests was carried out to assess the performance of the simulator .

The test results obtained can be split into two separate parts to deal with the numerical results obtained and the time taken to simulate different sizes of system :

First, a set of tests was run to compare the numerical results of the parallel algorithm with those produced by a simulator used by the research group at Durham . This simulator uses a Newton-Raphson approach to solve the network equations and an implicit trapezoidal technique to solve the differential equations . It was run with a one second time step to produce results in real time on a Perkin-Elmer 3230 mini computer . The parallel algorithm was run in real time with a 20 ms time step so that both the transient and steady state results could be compared .

Secondly, a series of tests was carried out on the M68020 system, using the floating point hardware, to produce timings for the routines used by the simulator for different sizes of network . These timings were then combined to calculate the number of slave processors that would be required to run the simulator in real time for various sizes of network at different time steps .

The test system was based upon the IEEE 30 substation test network which consists of 6 generators, 41 lines, 21 loads and 30 buses as shown

in figure 7.1 . The system parameters have been enlarged to include the parameters of the generators in the system which were not needed for a load flow solution but are essential for simulation . Also the inclusion of a varying load curve for the simulators means that the loads are different from those presented in the original test network data for load flow calculation . The load curve varies the load present in the system depending on the time of day . The simulators were started from an initial set of data corresponding to a midnight load pattern: this load pattern and the rest of the system parameters are presented in appendix 2 .

7.1 NUMERICAL RESULTS

Four different tests were performed to compare the numerical accuracy of the two simulators: all the tests were started using the midnight load pattern from the load curve . The first three tests were short term and involved the altering of load,line and generator parameters respectively . The fourth test was over a longer period of time and covered several changes in the system . This final test included the dividing of the system into first two and then three totally separate islands .

All four tests were carried out without any control action from outside the simulator after the alterations were made to the system . Thus no attempt was made to correct any frequency errors by load frequency control and no generator rescheduling or load shedding was performed . The system was allowed to settle down to steady state according to its internal control alone . The only control elements active in the simulator

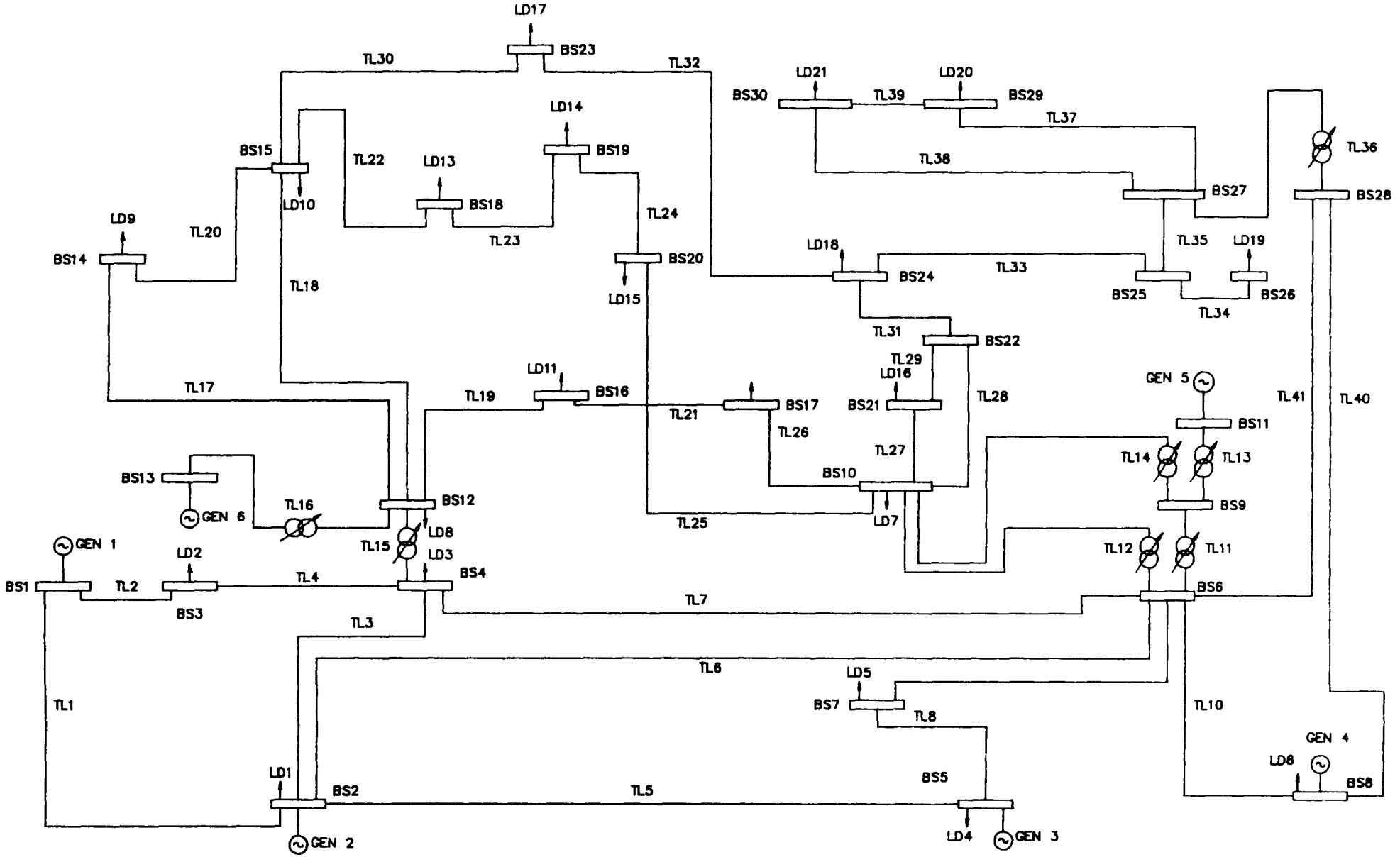


Figure 7.1) IEEE 30 bus test network

are the governors on the turbines driving the generators . These governors vary the mechanical power input to the generators depending on the power and frequency deviations from the set points .

The tests were run with all the loads in the system represented as part of the network (constant impedances to ground): this meant that both line outages and load changes required the Householder routine to be run .

7.1.1 LOAD OUTAGE TEST

The first test consisted of the removal of load number 4 . This load represents almost a third of the active load in the system and more than a seventh of the reactive load . Thus its removal should have quite a large effect on both the frequency and voltage of the system .

With the removal of a large amount of reactive load in the system the voltage magnitude throughout the system should rise . Figure 7.2 shows the voltages at the bus containing load 4 (bus 5) and another bus well away from load 4 (bus 28) . The graphs show the load outage occurring after 3 seconds and both show the rise in voltage expected, bus 5 being effected significantly more than bus 28 because of its proximity to generator 3 . The graphs also show the similarity between the numerical results obtained from the two simulators .

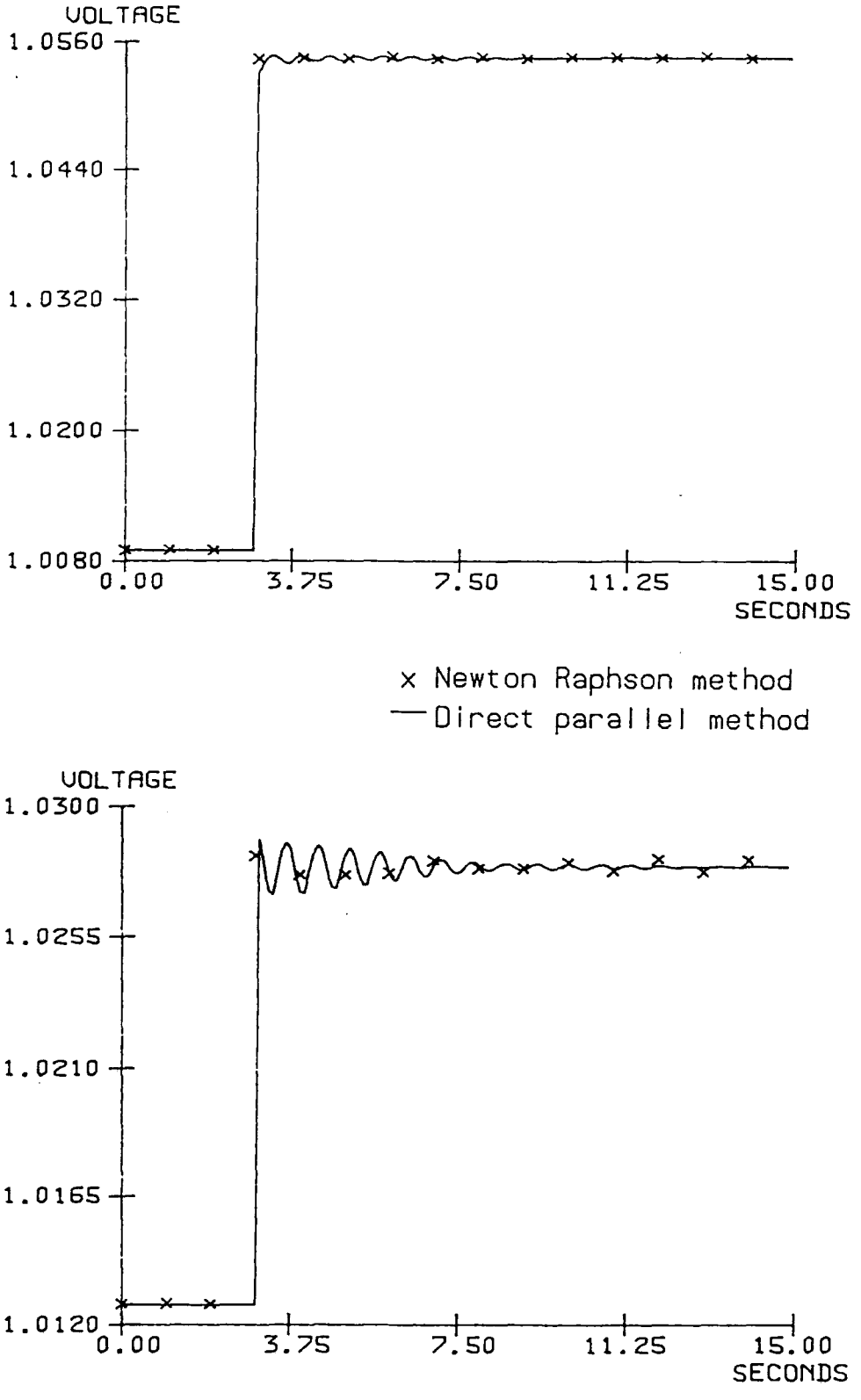


Figure 7.2) Voltages at bus 5 (top) and 28 during test 1

The removal of active load from the system should cause a change in the system frequency . Because no control action is being taken (apart from the governor on the turbine) the rise in the steady state frequency can be estimated . From the equation defining the governor response (4.1.3) and looking for the steady state frequency then :

in steady state

$$\frac{dP_m}{dt} = 0 \qquad (7.1.1)$$

into (4.1.3)

$$0 = P_{set} - P_m + K * (F_{set} - \omega/2\pi) \qquad (7.1.2)$$

and hence

$$P_{set} - P_m = K * (\omega/2\pi - F_{set}) \qquad (7.1.3)$$

Because all the generators in the system have a gain of 1 and there are six generators in the system with governors on then the left hand side of equation (7.1.3) can be set equal to one sixth of the active power lost to estimate the new steady state frequency :

$$0.132 = (w/2\pi - 50.00) \qquad (7.1.4)$$

The value of $\omega/2\pi$ thus produced ($\omega/2\pi$ = 50.132) is the estimated frequency after the load has been tripped . This assumes that the rest of the load remains constant which is not quite true; because the loads are represented as constant impedances to ground

then the rise in system voltage after the load outage causes a rise in the total load in the system . Thus the active load lost is less than the active load shed by the outage and so the frequency rise given by equation (7.1.4) will be an over estimate .

Figures 7.3 to 7.5 show the speed and power produced by three of the generators, these all show that the per unit speed of the system after the load trip increases to approximately 1.0024 . Since the per unit frequency base for the system is 50.0 Hz this corresponds to a frequency of 50.12 Hz which agrees with the estimate given above .

The powers shown in figures 7.3 to 7.5 also show that the reduction in generated power is shared equally among the generators: any variation in the governor gain between generators will distribute the power reduction unequally, but with all the governor gains set to 1.0 the power drop is distributed equally .

The graphs also show that although the steady state values for both simulators are the same the transients are quite different . The parallel simulator using a much shorter time step shows the oscillations of the individual generators and the interaction between different generators . For example figure 7.4 clearly shows two sets of oscillations superimposed on one another, the faster oscillations of the generator have a period of less than a second, the longer ones have a period of four or five seconds and are caused by the interaction of the generators with different inertias and hence different response times .

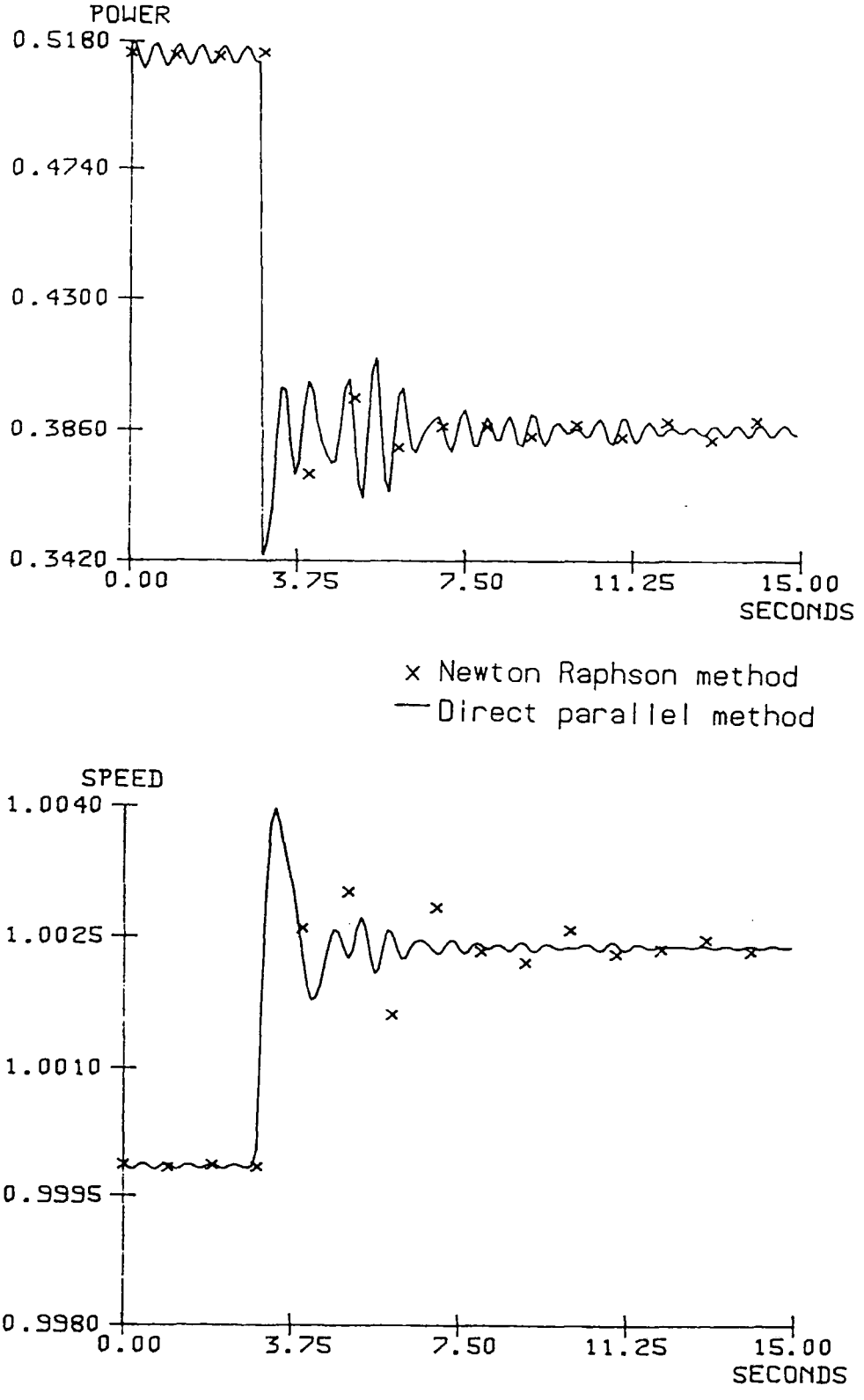


Figure 7.3) Power and speed of generator 1 during test 1

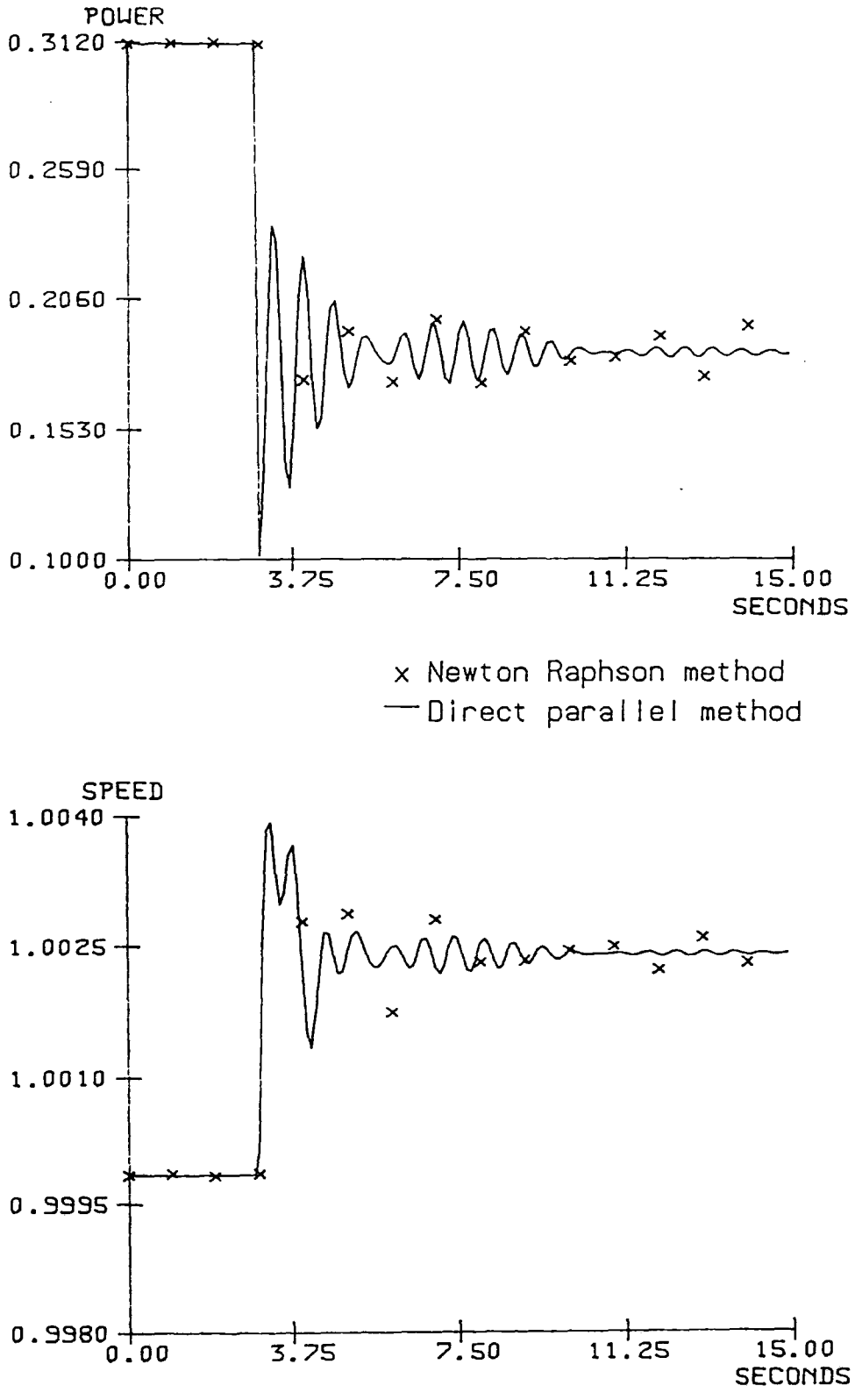


Figure 7.4) Power and speed of generator 3 during test 1

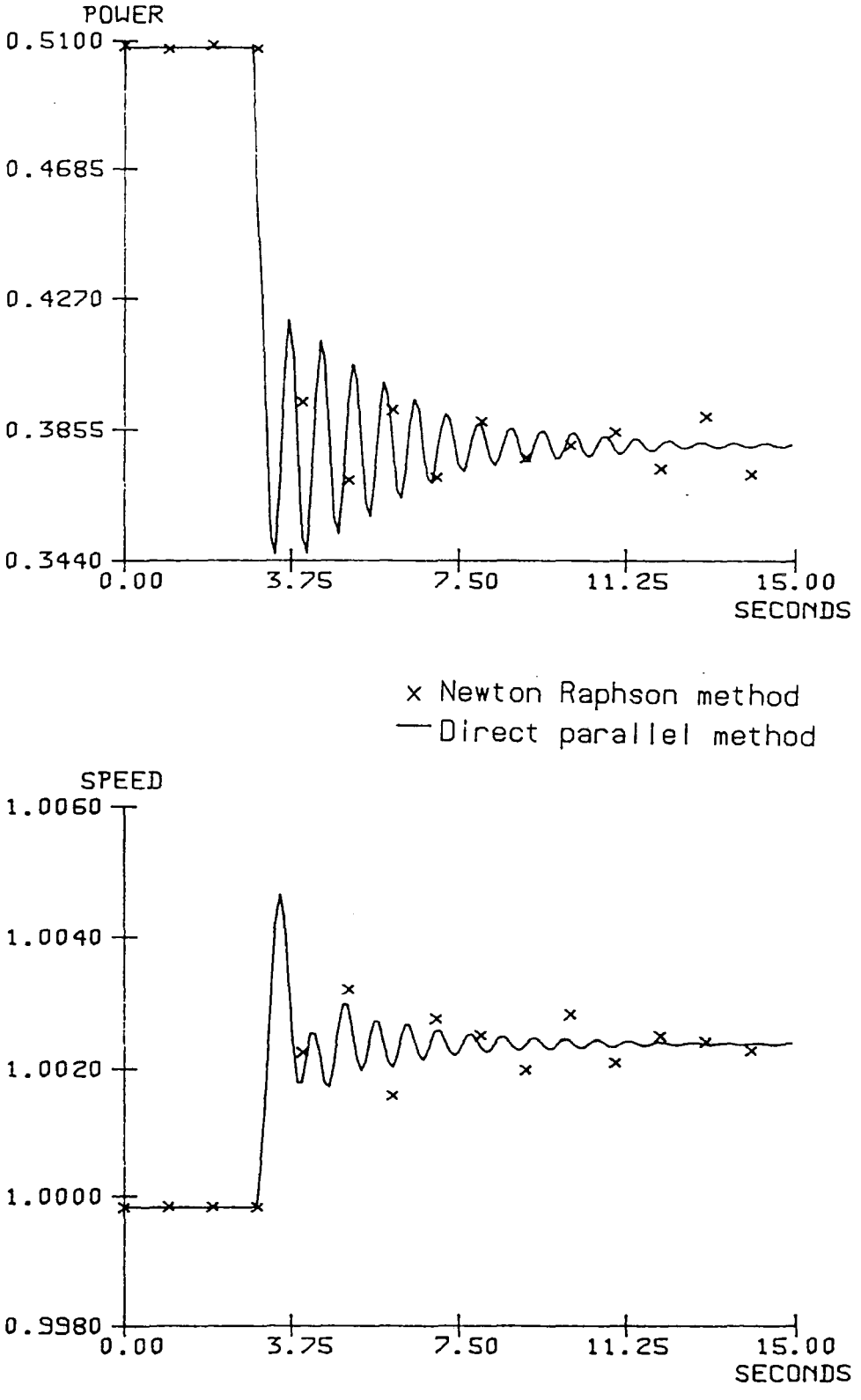


Figure 7.5) Power and speed of generator 5 during test 1

The Newton-Raphson simulator cannot show these short term effects because of the one second time step needed to run the simulator in real-time . Also the generator model using a one second time step is not as accurate during transients as the parallel model using a 20 ms time step: this inaccuracy causes the oscillations to carry on far longer than they should . However this does not affect the steady state results produced by the simulator once the oscillations have been damped out . As a further test, the time step on the Newton-Raphson simulator was decreased to 20 ms and the test rerun: the oscillations on this second run damped out much faster but the steady state results produced were the same as with the longer time step .

7.1.2 LINE OUTAGE TEST

The second test deals with the removal of lines from the network . Two lines were outaged, one after the other, and the lines were chosen so that when both are outaged one of the buses, which had been tightly coupled to generation, becomes remote . The first line outaged is line 40 and then, two seconds later, line 41 is also removed . This causes bus 28 to become remote from all the generation, having been closely connected to generator 4.

Figure 7.6 shows the effect of the outages on the voltage magnitude at two of the buses in the system, the first outage

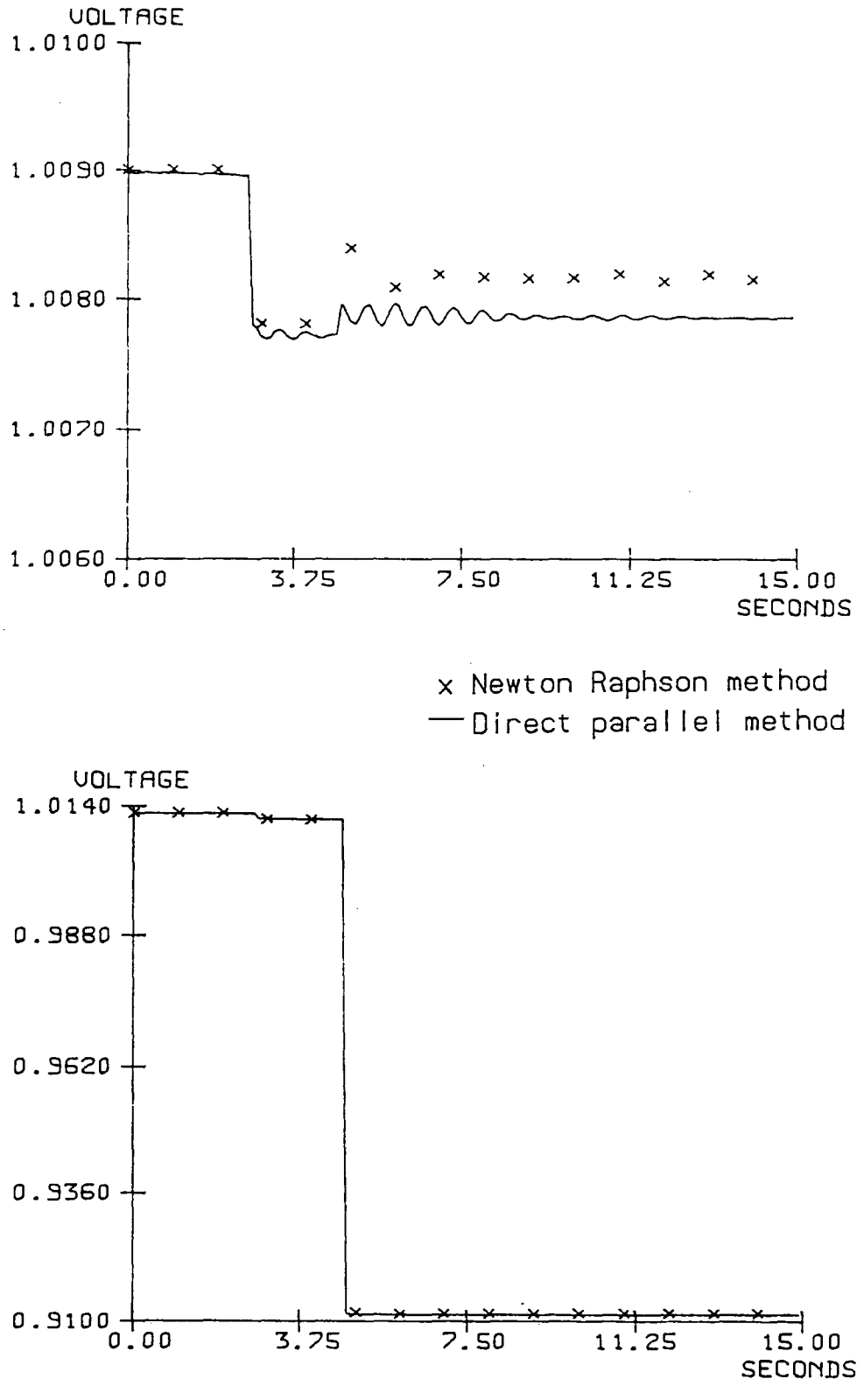


Figure 7.6) Voltages at bus 5 (top) and 28 during test 2

occurring after three seconds and the second after a further two seconds .

The first outage causes the voltage at both bus 28 and bus 5 to drop slightly: this is because the power finds alternative routes to bus 28 and the loads 19, 20 and 21 which it helps supply . The alternative routes involve a slightly higher amount of line loss in the system so the voltage drops throughout the system .

The second outage causes a much larger drop in the voltage at bus 28 . The bus has now become remote and has no power flow through it: the transformer between bus 28 and bus 27 (line 36) serves no purpose now except maintaining the voltage at bus 28 . The voltage at bus 5 rises when the second line is outaged . Once again the power flow has found an alternative path to loads 19, 20 and 21 and the line losses have increased . However, the voltages at buses 27, 29 and 30 have dropped along with bus 28 so the power of the loads in that area has also dropped . The overall effect is a reduction in the total system load, so the voltage at bus 5 increases . The difference between the voltages produced by the two simulators for bus 5 is very small, approximately 0.0004 per unit . This is due to the convergence on the Newton-Raphson simulator being set to 0.005 per unit so the results are well within the convergence .

Figures 7.7 to 7.9 show the power and speed levels of three of the generators during the line outages . The power drops and the frequency rises for both outages: this is caused by the load decrease brought about by the change in voltage levels . The effect on both

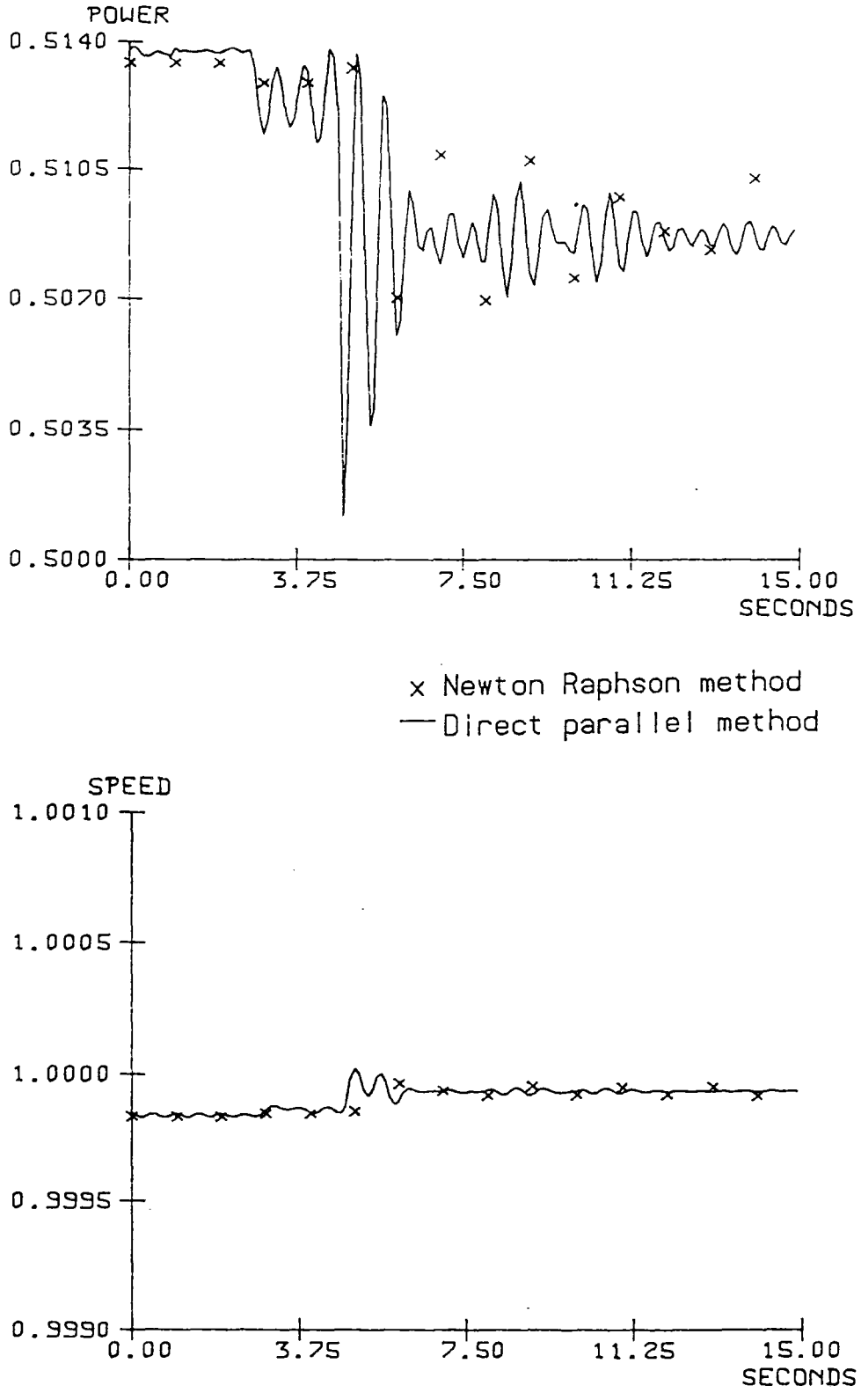


Figure 7.7) Power and speed of generator 1 during test 2

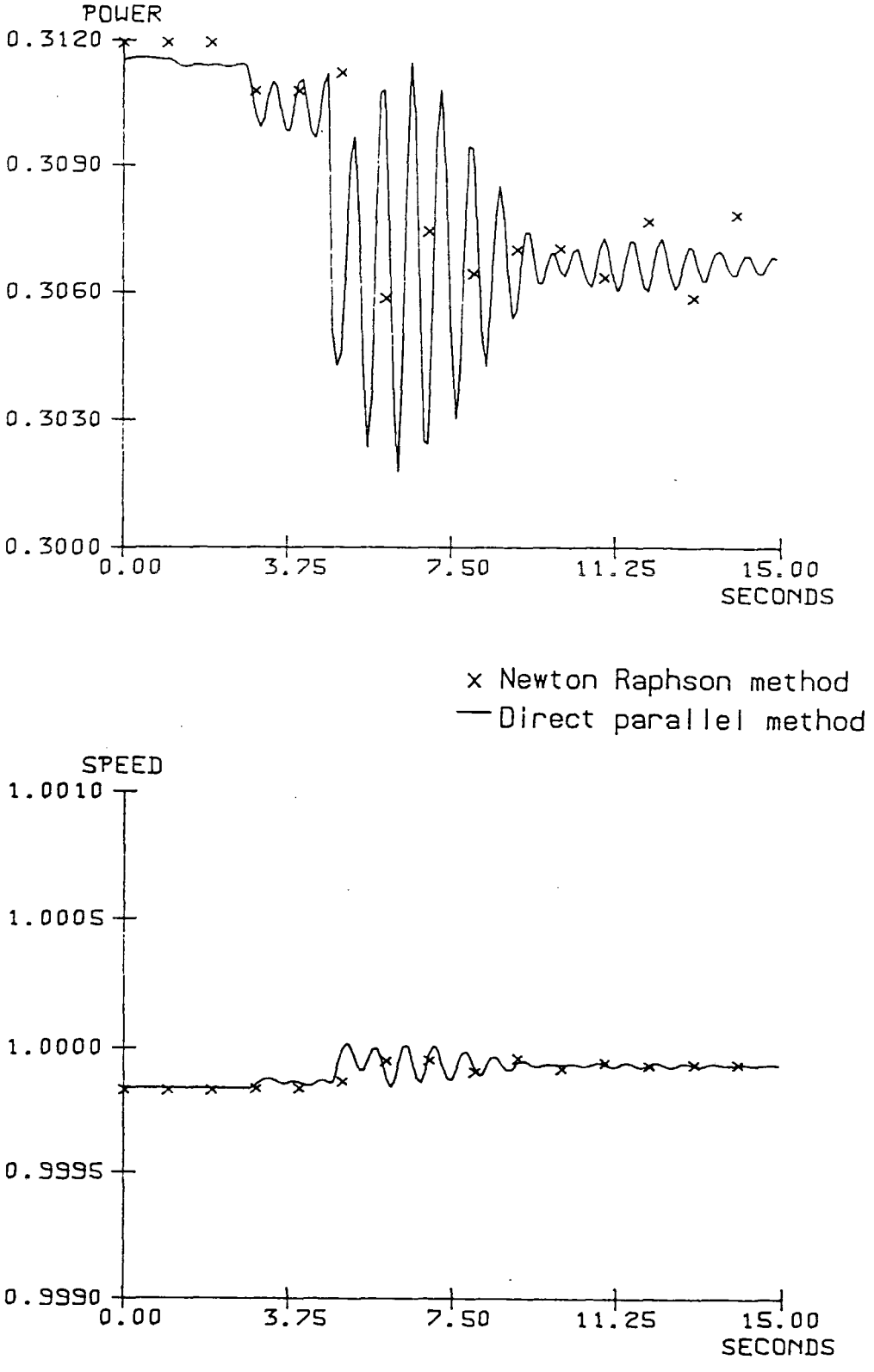


Figure 7.8) Power and speed of generator 3 during test 2

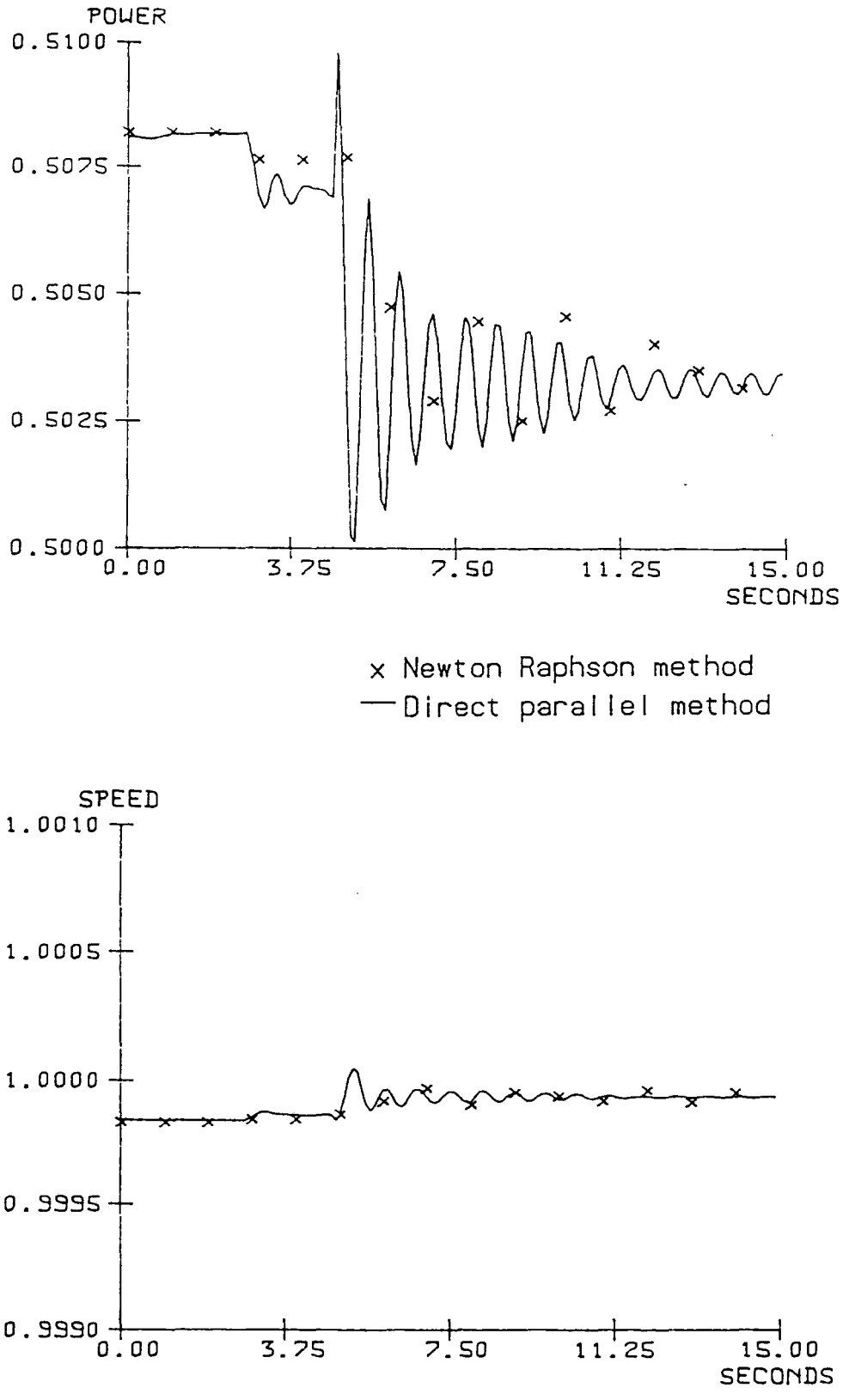


Figure 7.9) Power and speed of generator 5 during test 2

power and speed is far smaller than that caused by the load outage but once again the parallel algorithm shows the two frequencies of oscillation superimposed .

7.1.3 GENERATOR OUTAGE TEST

The third test involved the outage of one of the generators . The generator chosen for the test was generator 6: although this generates the smallest amount of active power it also generates the most reactive power . Because the voltage magnitude of the system is largely dependent upon the reactive power, the removal of this generator should have a large effect on both the system frequency and voltage .

Figure 7.10 shows the voltages at buses 5 and 28 . Both bus voltages drop sharply when the generator is removed after three seconds because of the loss of generated reactive power in the system .

Figures 7.11 to 7.13 show the speed and power for three of the generators left in the system . Although a generator has been removed from the system the power produced by the other generators in the system drops quite sharply and the speed increases . This is due to the fact that generator 6 produced a lot of reactive power while it was connected . When the outage occurs the loss of reactive generation causes the voltages around the system to drop sharply and this in turn reduces the system load . In this test the reduction in

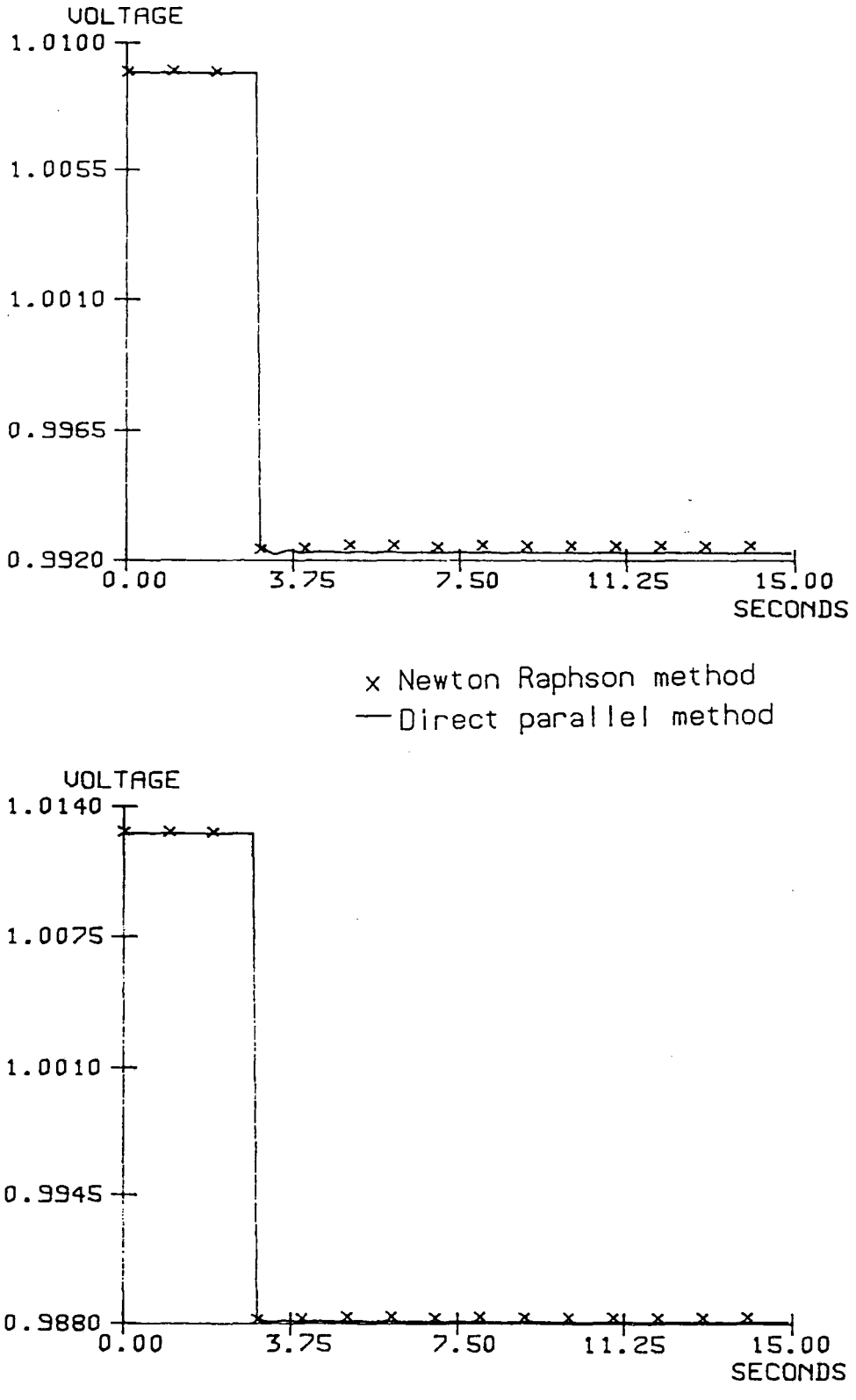


Figure 7.10) Voltages at bus 5 (top) and 28 during test 3

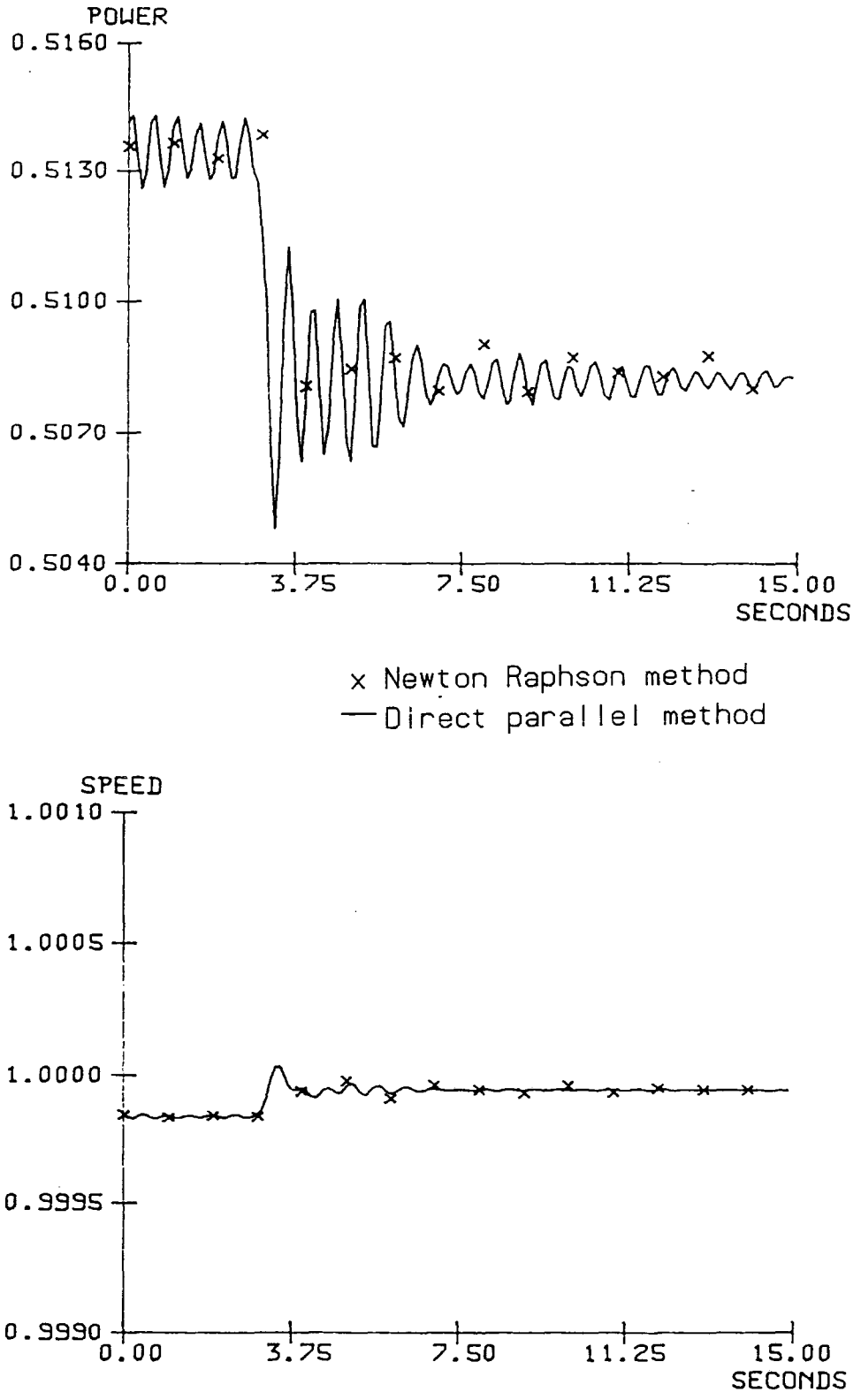


Figure 7.11) Power and speed of generator 1 during test 3

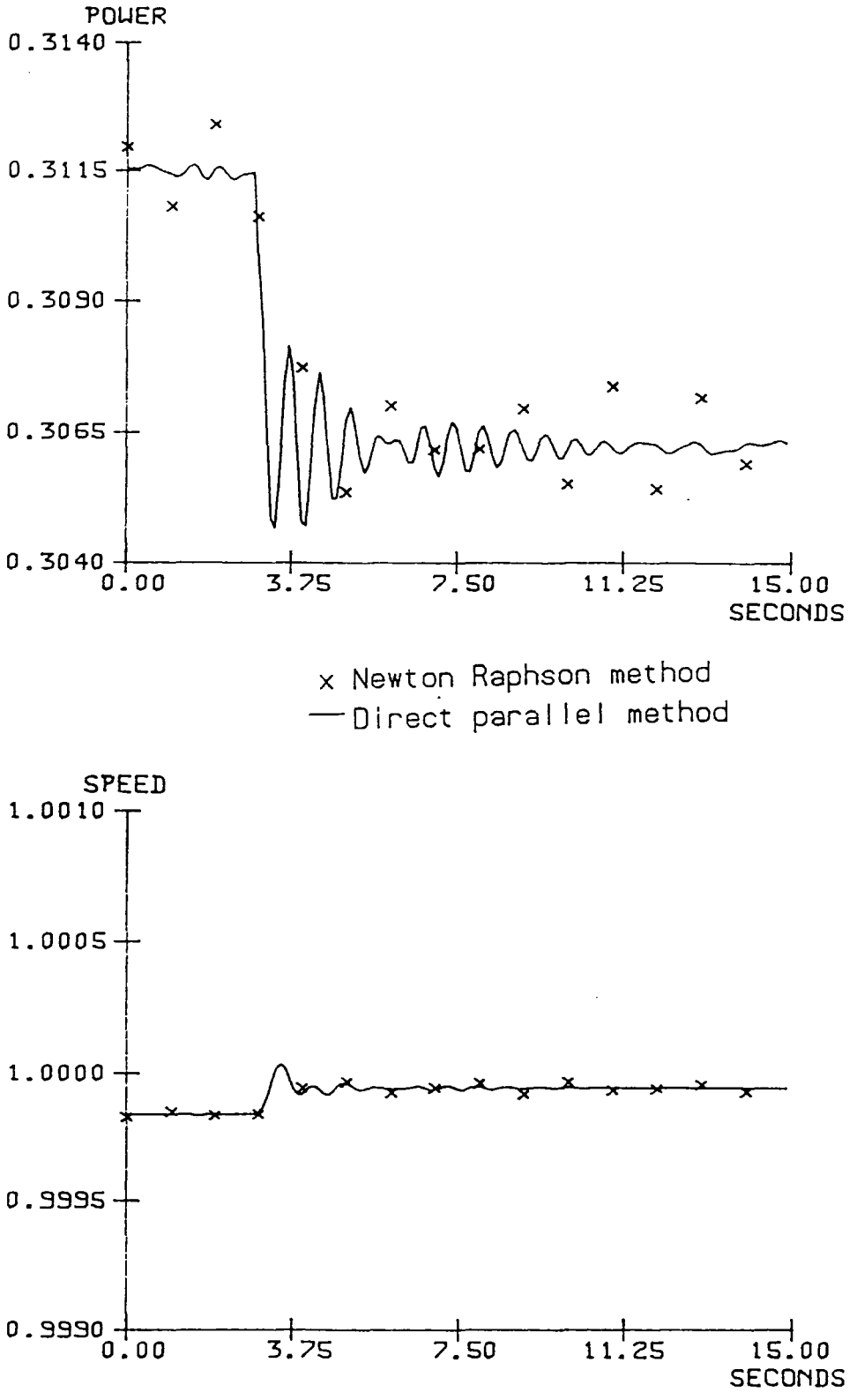


Figure 7.12) Power and speed of generator 3 during test 3

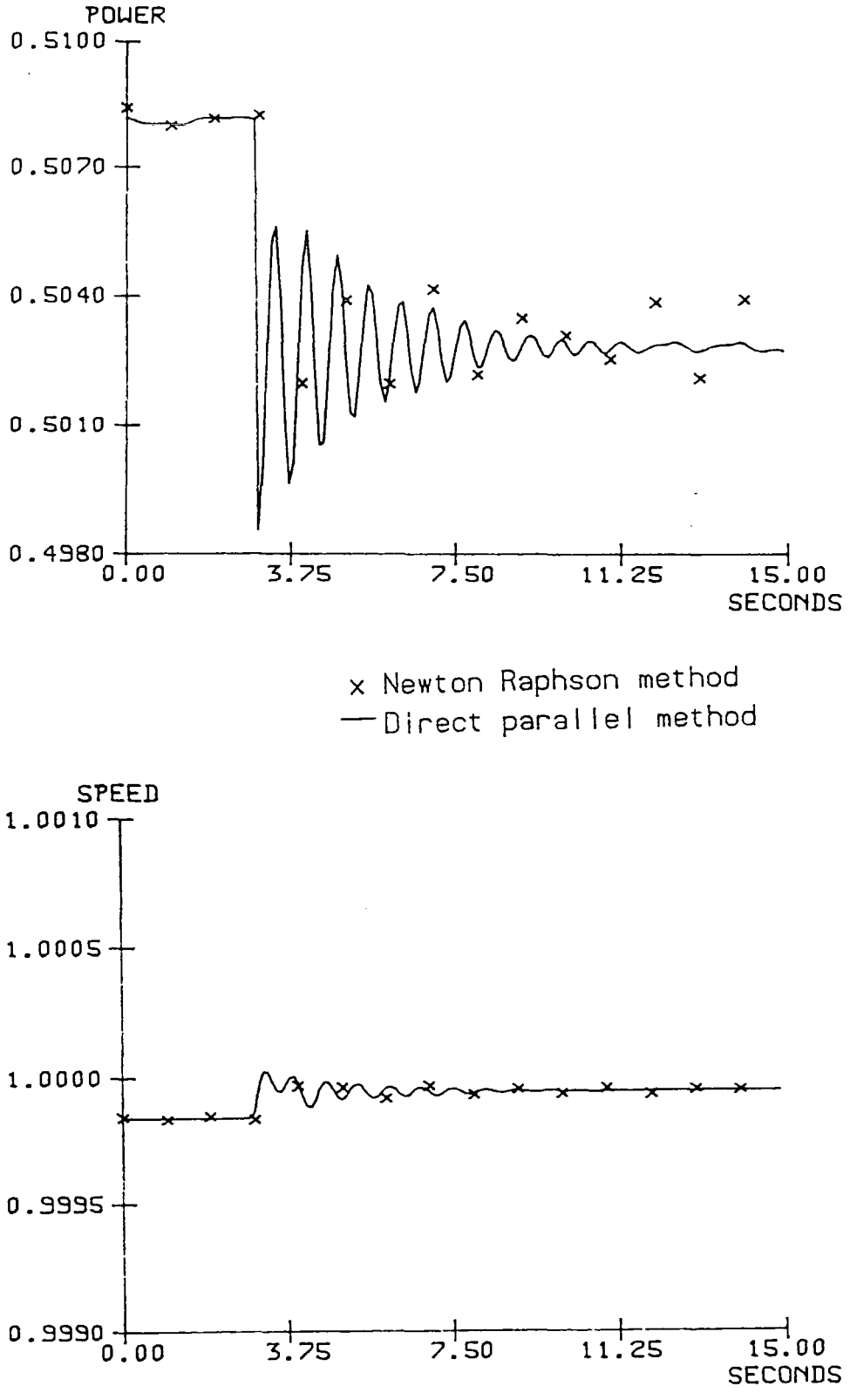


Figure 7.13) Power and speed of generator 5 during test 3

the active load power is greater than the loss in the generated active power so the governors (which respond only to active power imbalances) reduce the active power slightly so a balance is achieved .

The resultant changes in the speed and generated power are similar to the load outage test, but the voltage response of the system is very different . This test clearly shows the large coupling between reactive power and voltage magnitude which, along with the active power - voltage angle coupling, as used in the decoupled load flow technique .

7.1.4 MULTIPLE EVENT TEST

The fourth test carried out was over a longer period than the first three tests and involved a series of events as set out in table 7.1 . The purpose of this test was to look at a series of changes to the system and finish by splitting the system into a number of totally separate islands . Some network solution methods develop problems as soon as the system splits into islands, but the direct method used for the parallel simulator continues and calculates the simulation for all the islands . This simulation of islands is entirely independent of the way the system is split into parts for solution on the parallel processors .

The sequence of events is run on a simple scenario generator which controls the simulator and causes the events at the required

times . Figure 7.14 gives the voltage for two of the buses in the network . These show small variations in voltage for the events up to number 8 (i.e. the removal of line 10) and then, after the system islands the events only have an effect on certain parts of the system . Figures 7.15 to 7.17 give the generator responses for one generator in each of the three islands formed at the end of the test . These show that the three generators respond in much the same manner until the system becomes islanded and they become separated from one another .

EVENT NUMBER	TIME (SECS)	EVENT
1	30	REMOVE LINE 41
2	50	REPLACE LINE 41
3	60	REMOVE LINE 24
4	70	REMOVE LINE 15
5	75	REMOVE LOAD 14
6	95	REMOVE LINE 19
7	100	REMOVE LINE 41
8	105	REMOVE LINE 10
9	120	REPLACE LOAD 14
10	130	REMOVE LINE 33(system 2 islands)
11	150	REMOVE LINE 32(system 3 islands)
12	155	REMOVE GENERATOR 6
13	160	REMOVE LOAD 4

Table 7.1 Events for test 4

For comparison, the speeds of these generators are shown together in figure 7.18 . After the islanding, generator 6 is removed: this causes one of the islands to have no generation and the voltages in that island drop to zero . Generator 6 continues to try to generate power because no new power and frequency set points are

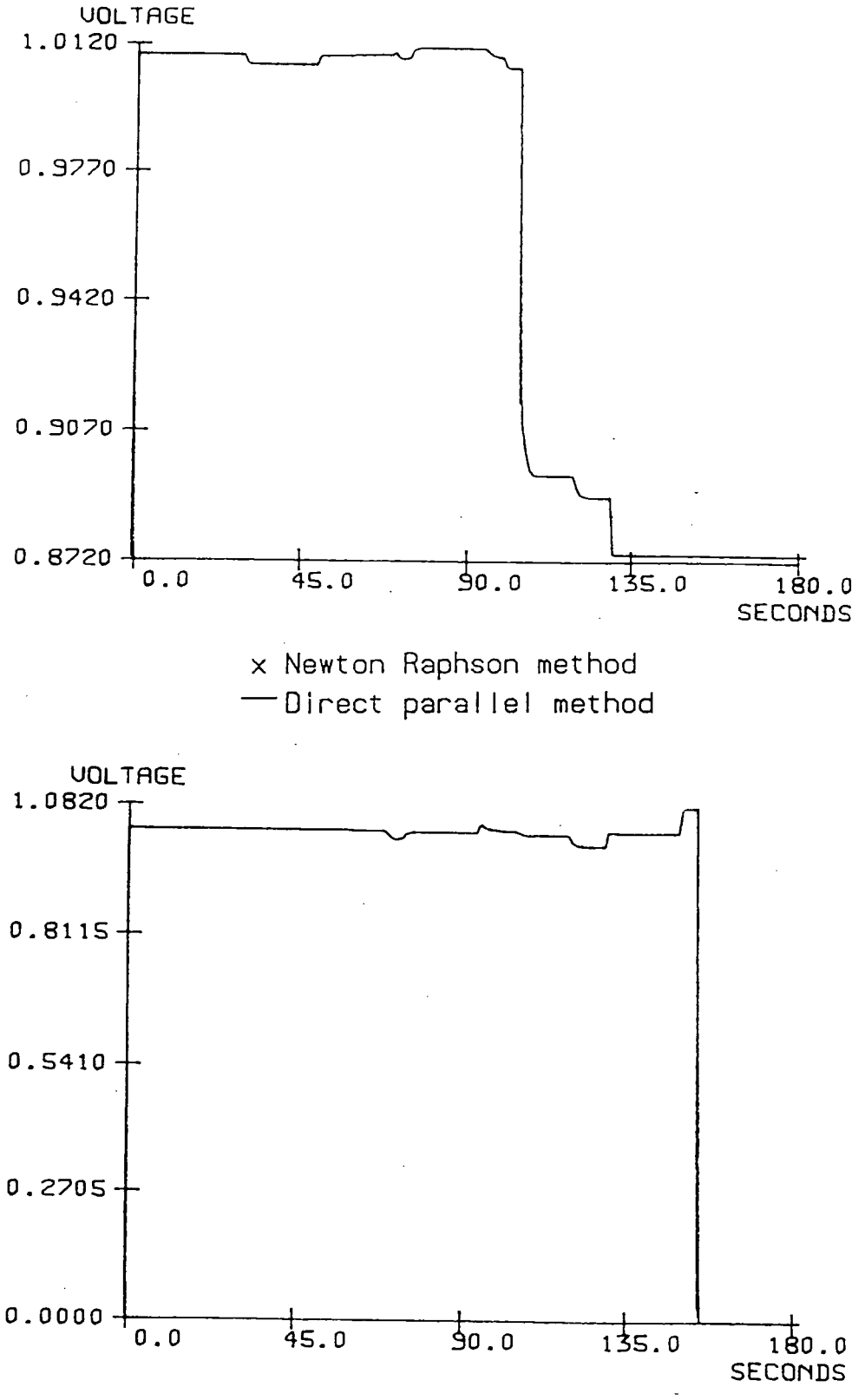


Figure 7.14) Voltages at bus 5 (top) and 28 during test 4

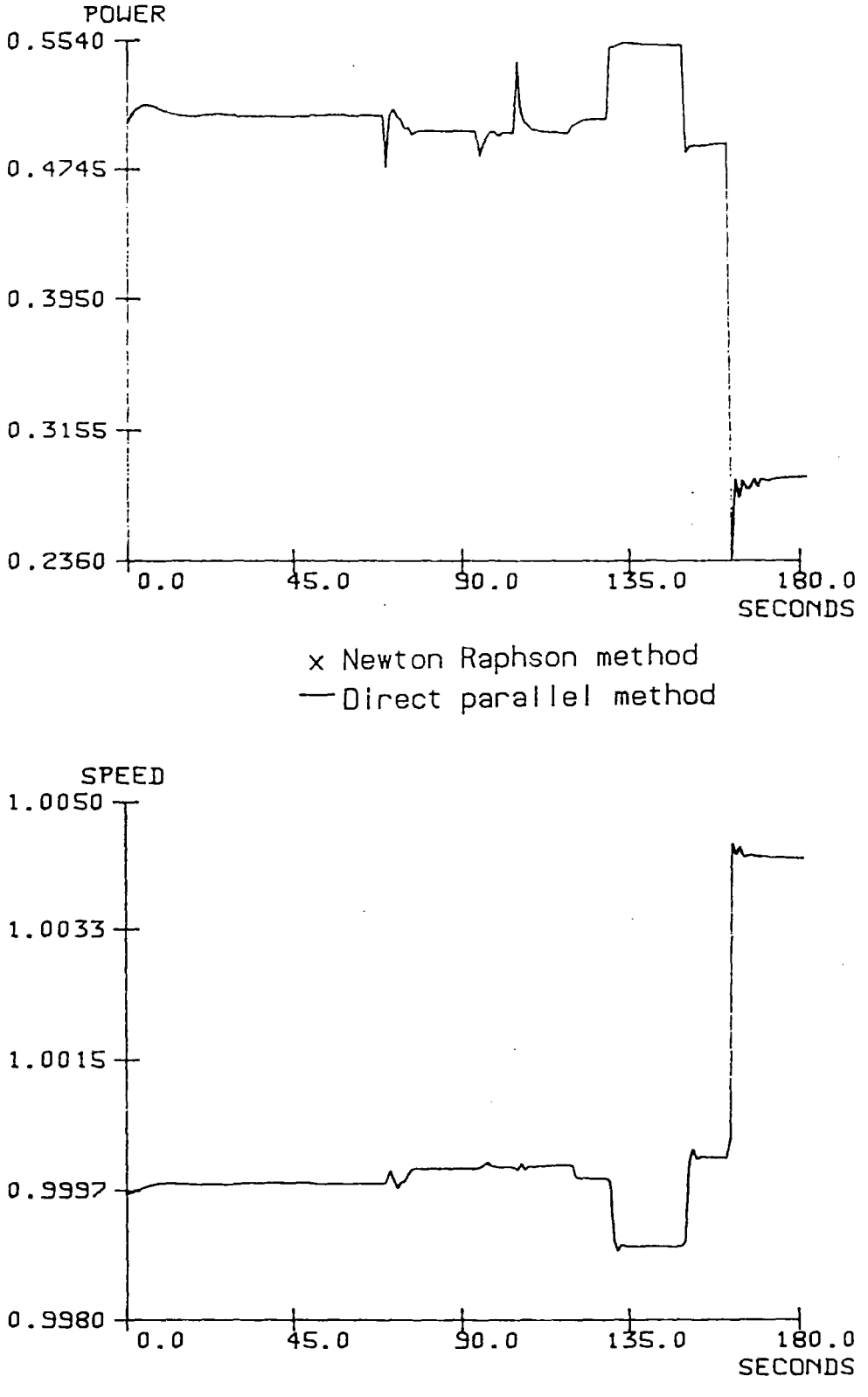


Figure 7.15) Power and speed of generator 2 during test 4

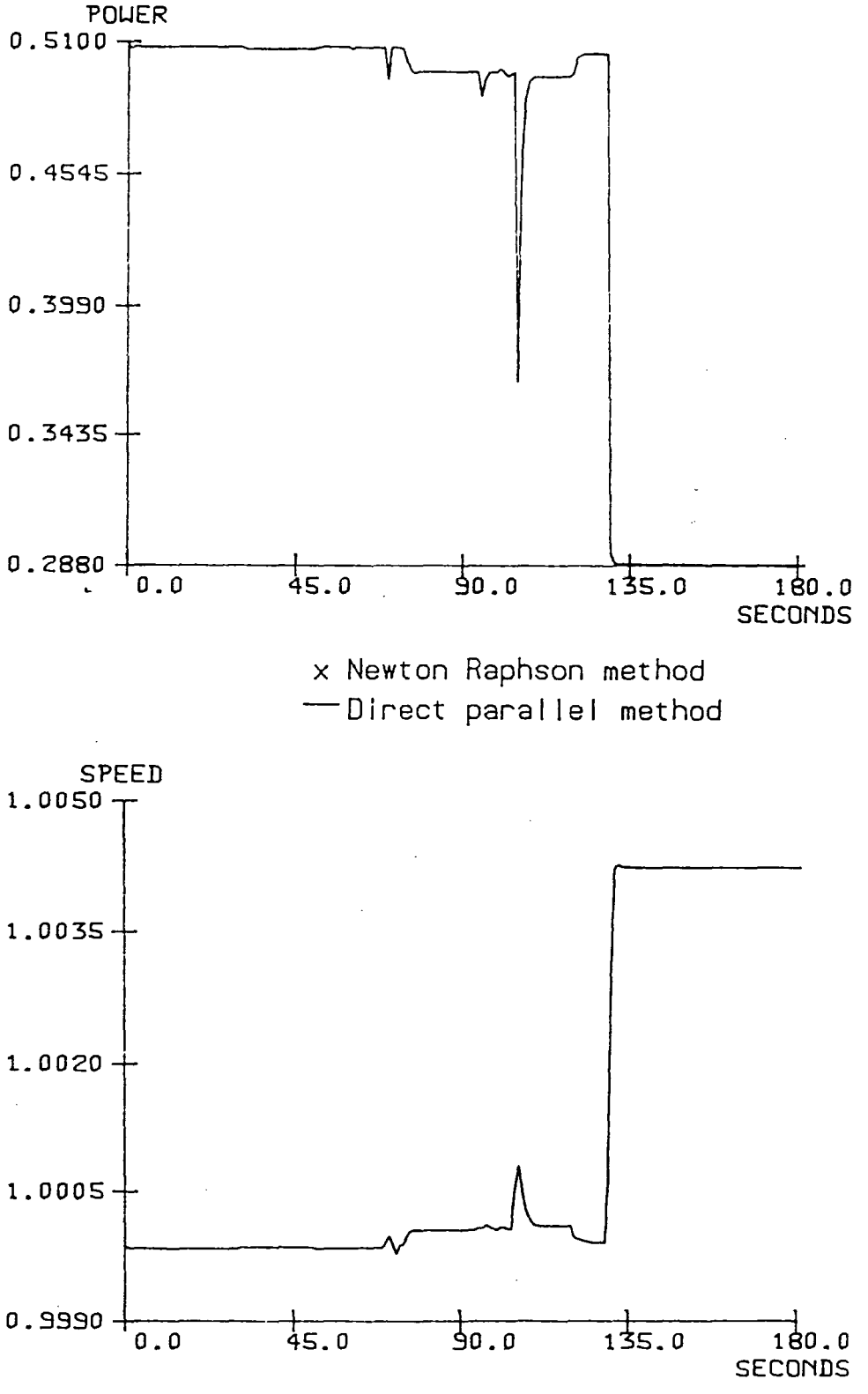


Figure 7.16) Power and speed of generator 4 during test 4

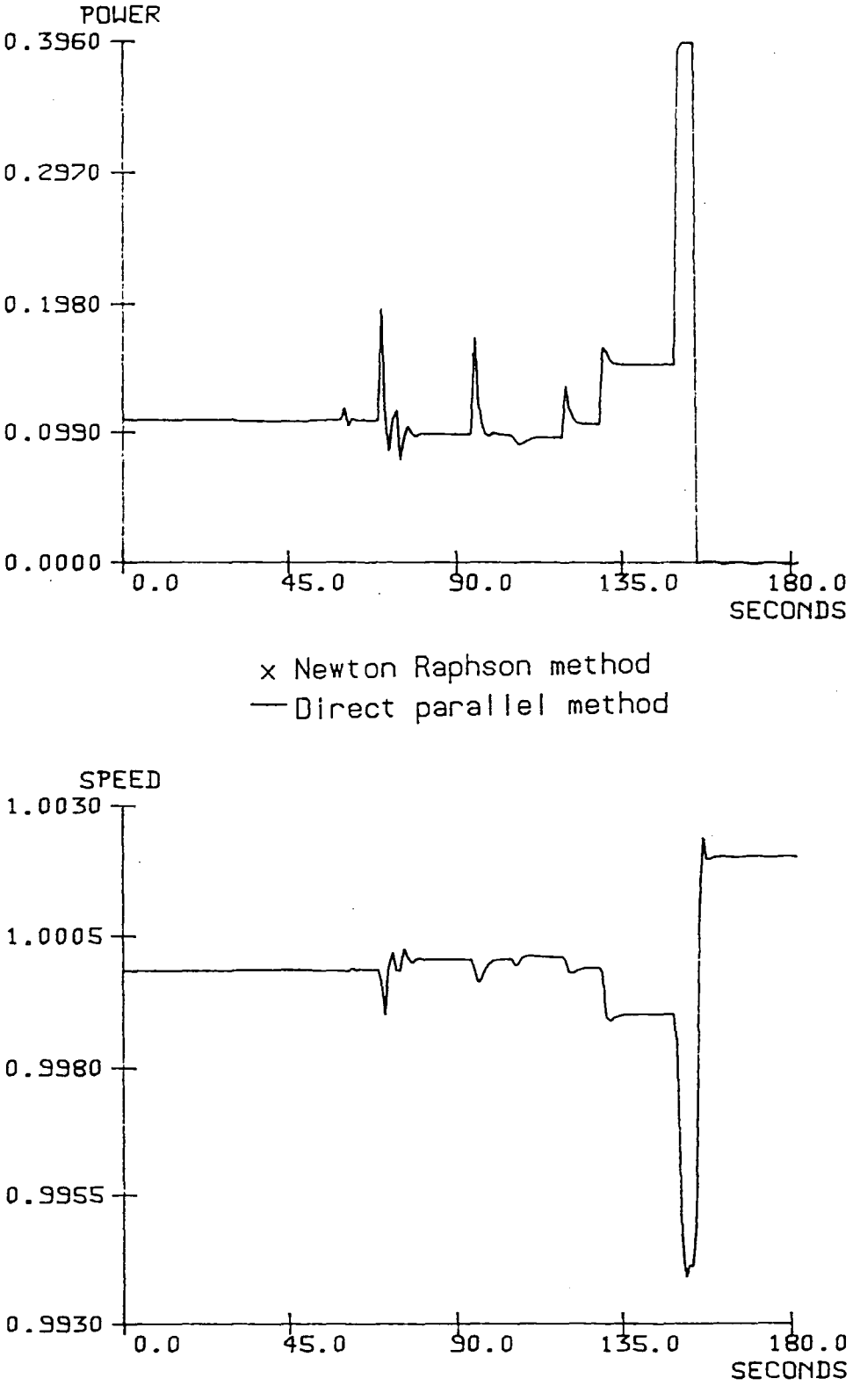


Figure 7.17) Power and speed of generator 6 during test 4

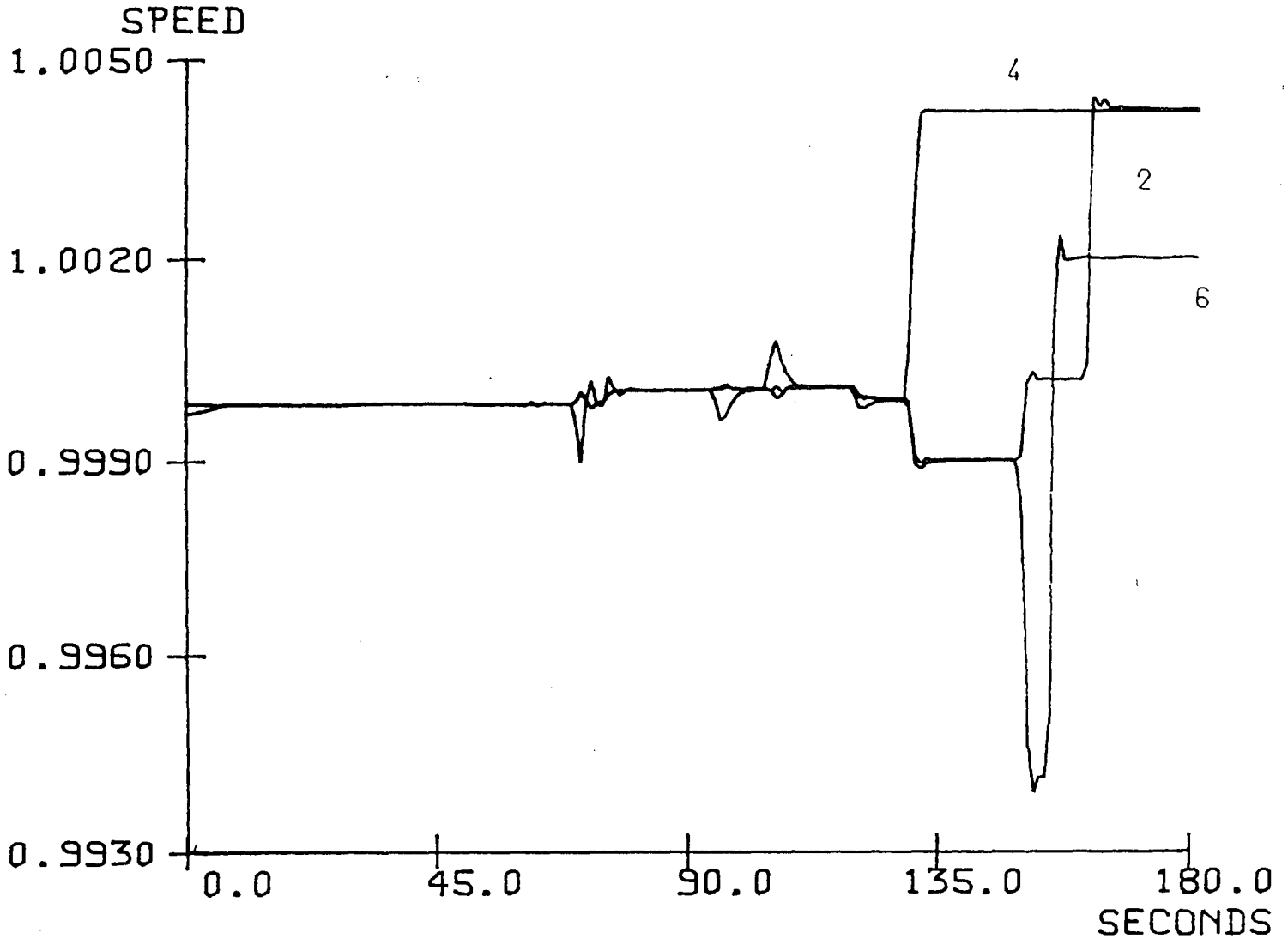


Figure 7.18) Speeds of generators 2,4 and 6 during test 4

sent to it . When load 4 is removed the speed in the remaining two islands moves together and control algorithms on the host could be used to synchronise the islands to bring the network back into one piece .

7.2 TIMING RESULTS

The second set of results were taken using the M68020 system with the Weitek floating point processor board . These results record the time taken by the various algorithms depending on the size of the system being simulated . All the algorithms using the hardware floating point board were written in 'C': some benchmarks were run on the hardware to compare it's speed with the minicomputers in use . The timings for the simulator are split into four parts, the generator routine, network routine, Householder routine and the interprocessor communication . The timings given for the software routines do not include any time for communication between processors .

7.2.1 GENERATOR AND LOAD ROUTINE TIMING

The timing for the generator routine to calculate the dynamic response of a single generator is constant for any network, regardless of the number of buses or generators simulated or the number of slave processors used . The second order generator with a first order governor equation as set out in chapter 4 ran in 0.28 ms,

this time was for one complete calculation, including both the initial estimate and final estimate pass through the generator routine . Thus the time spent calculating the generator dynamics for a single time step is :

$$T_g(\text{sec}) = \frac{0.28 * nga}{1000} \qquad (7.2.1)$$

where nga is the number of generators solved by a single slave processor . The timing for the type of static load representation given by equation (4.2.3) is also constant regardless of the network size . The routine runs in 0.1 ms, so if nla is the number of loads each area has to calculate :

$$T_l(\text{sec}) = \frac{0.1 * nla}{1000} \qquad (7.2.2)$$

7.2.2 NETWORK ROUTINE TIMING

The speed of the network routine is dependent on two factors: the number of generators (and loads not represented as constant impedances to ground) in the system and the number of bus voltages to be calculated by each slave processor . A series of timings were carried out varying these factors and the results are presented in figures 7.19-7.24 . These figures present the timings in graphical form: the actual values are presented in Appendix 3, as a set of points on graphs of speed against number of voltages calculated . Each graph is for a different number of generators in the system .

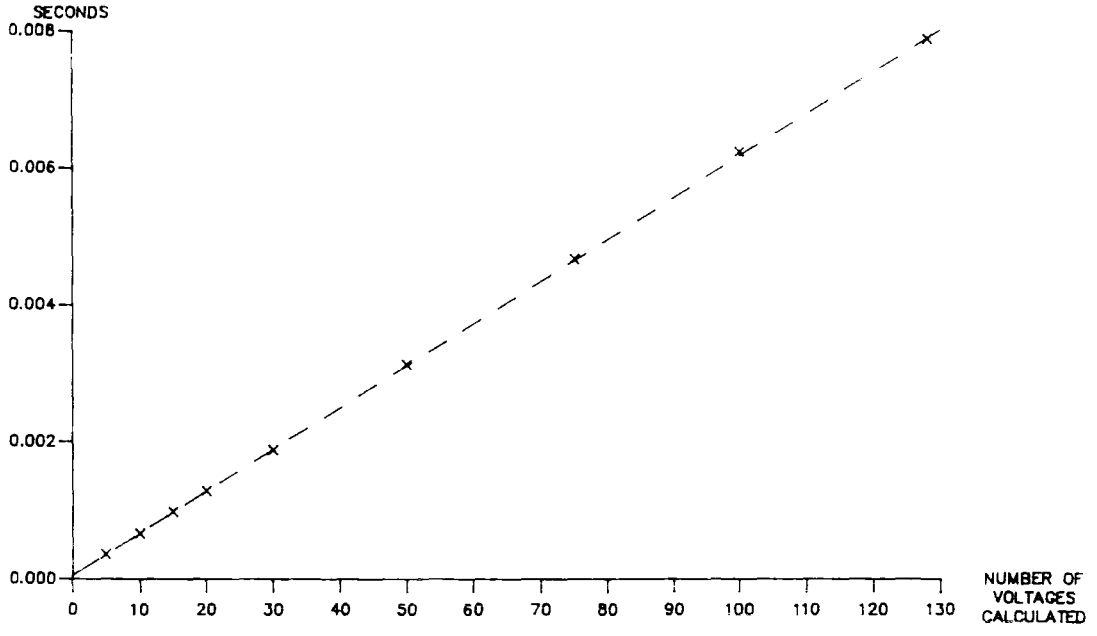


Figure 7.19) Network routine speed with 2 generators

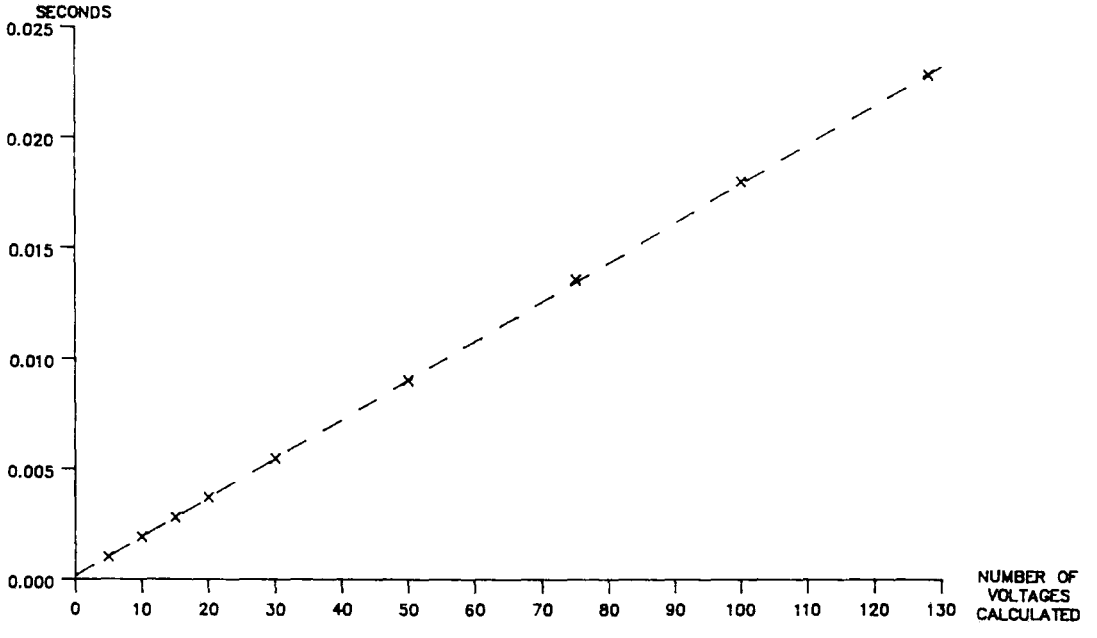


Figure 7.20) Network routine speed with 6 generators

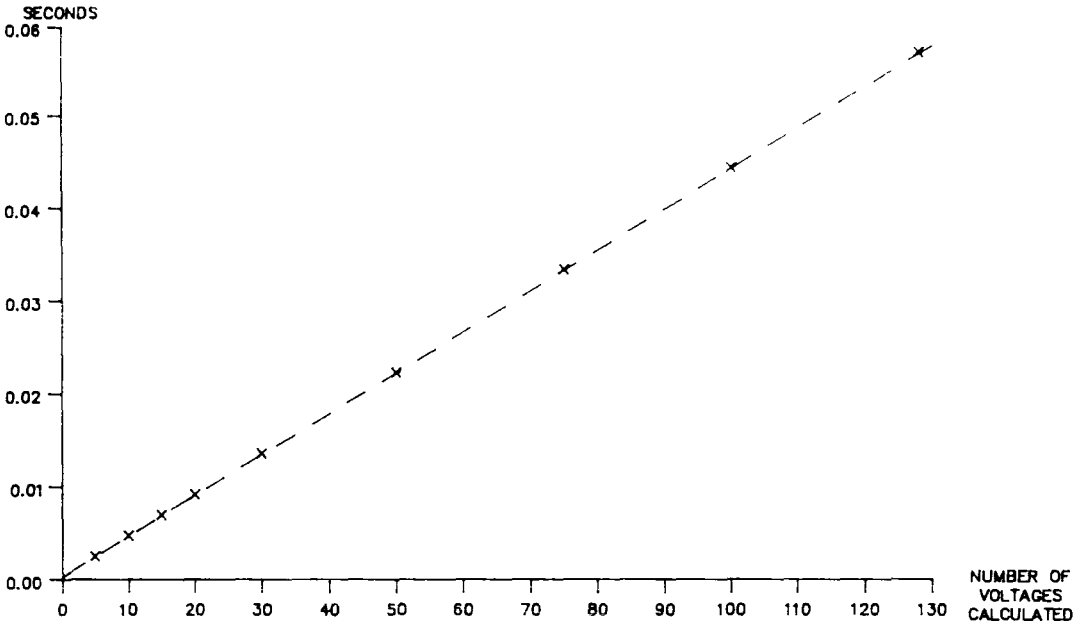


Figure 7.21) Network routine speed with 15 generators

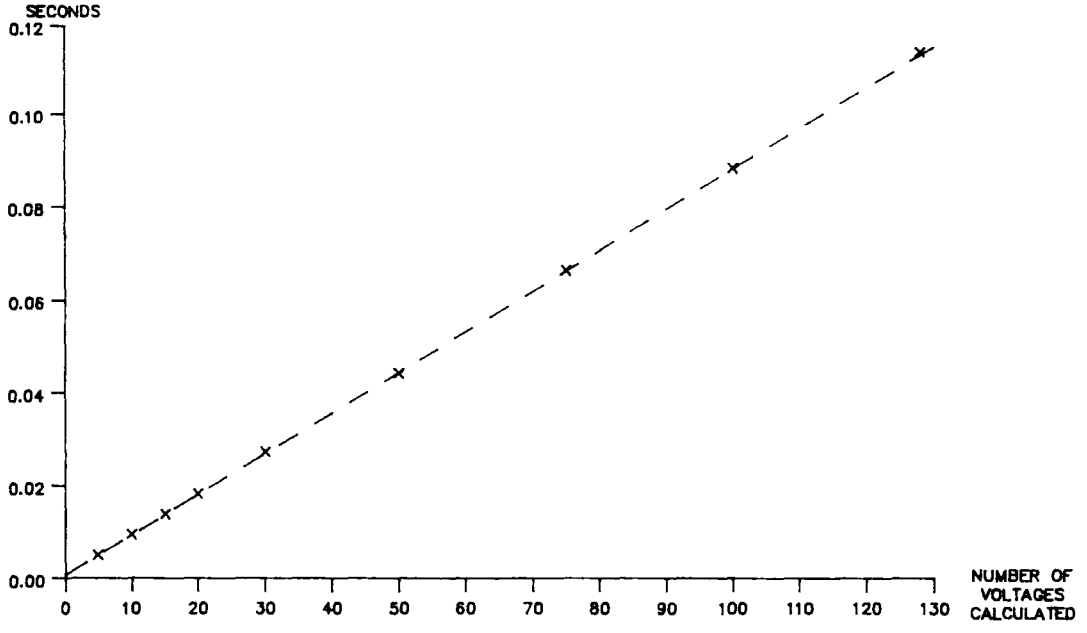


Figure 7.22) Network routine speed with 30 generators

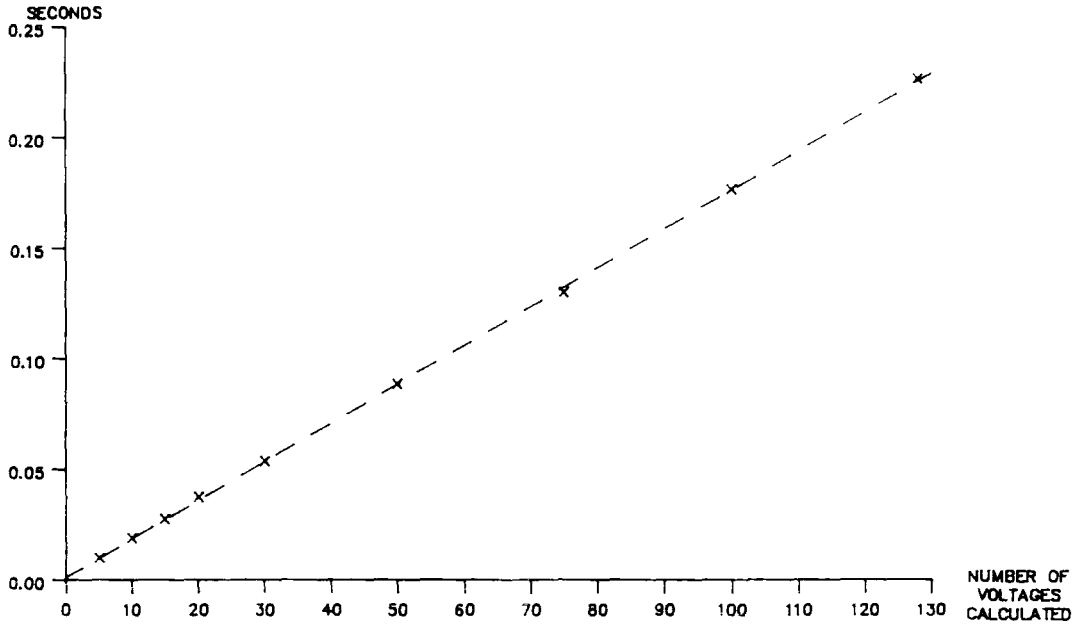


Figure 7.23) Network routine speed with 60 generators

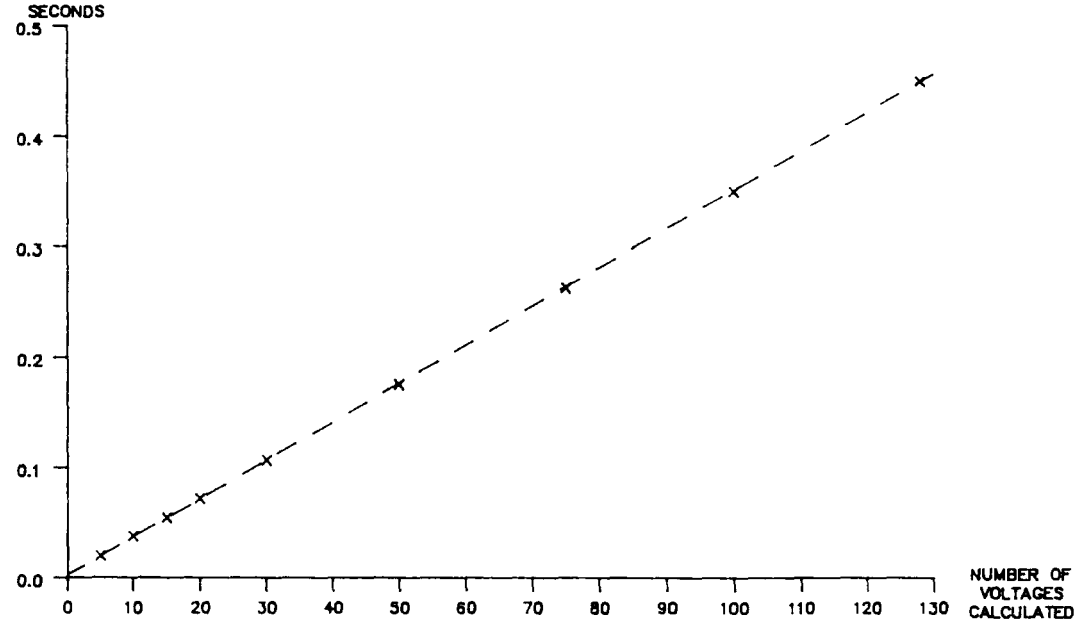


Figure 7.24) Network routine speed with 120 generators

The dotted line on each graph represents a straight line fit calculated by performing a fitting routine to the entire set of results . The equation represented by the lines is:

$$T_n(\text{sec}) = \frac{2.9087*ng*nba+0.2886*nba+1.8703*ng+2.1892}{100000} \qquad (7.2.3)$$

where ng is the number of generators in the system and nba is the number of bus voltages to be calculated in the area . If either of the variables is held constant this gives an equation of a straight line .

If the loads in the system are represented as current injections rather than constant impedances to ground the network time is increased . However, the increase is less than would result from including the loads in with the generators to calculate ng . This is because using currents instead of voltages saves a single complex multiplication in the inner loop of the network routine . Once loads are represented as current injections the generators can also be modelled in this way: this adds to the length of the generator routine but reduces the time spent in the network routine . Whether or not this saves time depends on the size of the network, but for networks larger than the IEEE 30 bus network this technique should be advantageous.

The equation corresponding to (7.2.2) but giving the time if all loads and generators are calculated as current injections is :

$$T_n(\text{sec}) = \frac{1.743*ngl*nba+0.2168*nba+1.135*ngl+2.046}{100000} \qquad (7.2.4)$$

where ngl is the number of generators plus the number of loads in the network . The generator routine now involves an extra complex divide (the longest of the floating point operations) and the load currents also have to be calculated . However by calculating the loads separately from the network, the need to use the Householder routine for load changes is removed and it is only used in network topology changes .

7.2.3 HOUSEHOLDER ROUTINE TIMING

The speed of the Householder routine is also dependent upon two parameters, the number of buses in the network and the number of buses contained in each area . A set of timings were carried out on the routine varying these parameters and the results plotted in figure 7.25 . These results are also presented in tabular form in Appendix 3 . The series of lines on this graph is again drawn using a straight line fit for all the points taken . Each line represents a fixed number of buses in each slave processor's area while the number of buses in the entire system is varied and plotted against the time taken by the routine . The equation of the straight line fit is:

$$T_H(\text{sec}) = \frac{1.9431*nb*nba+0.2696*nba+9.1326*nb+1.2671}{100000} \qquad (7.2.5)$$

where nb is the total number of buses in the system and nba is the number of buses in an area . The Householder routine is only run when a change occurs; either a change in load value if the loads are

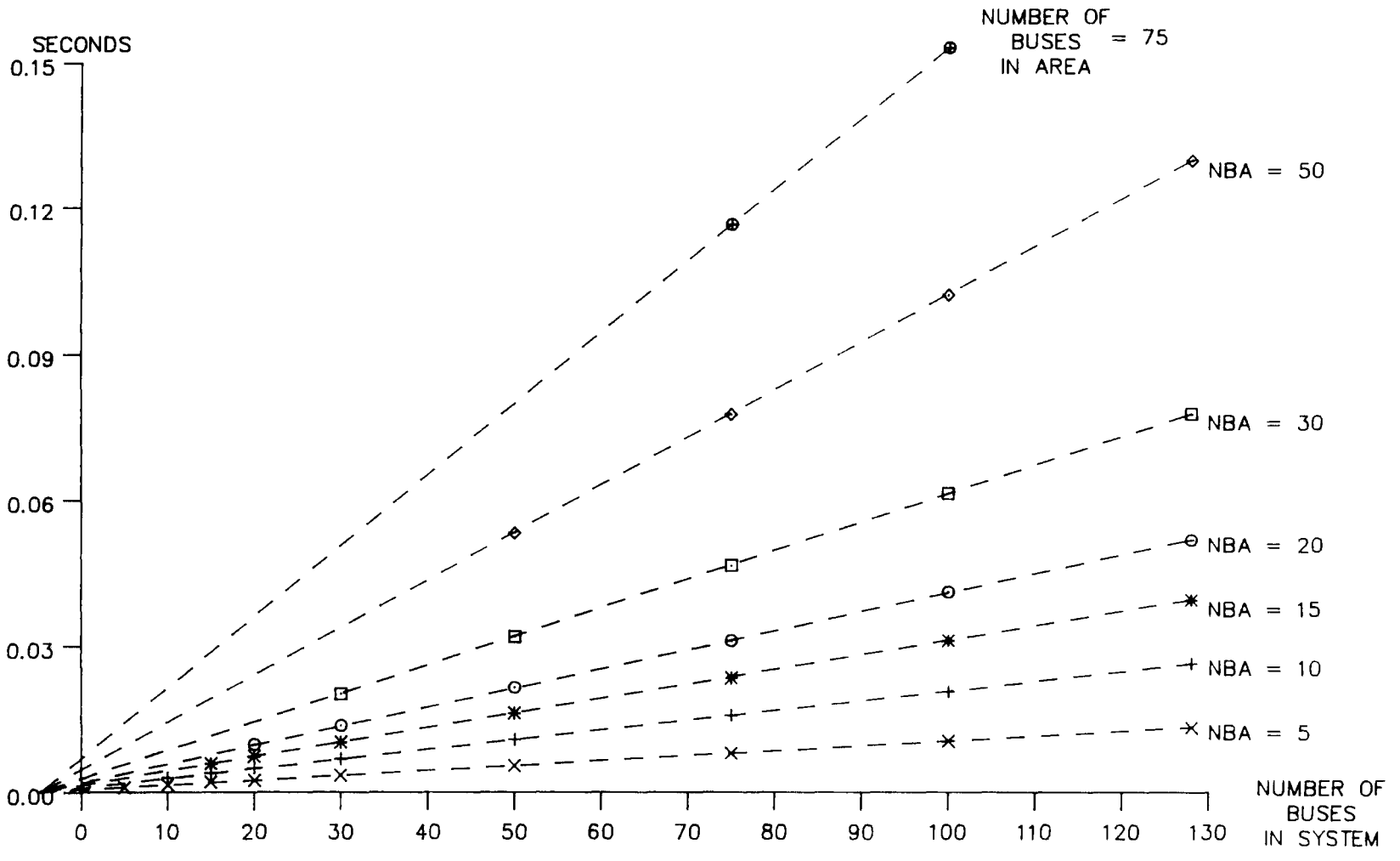


Figure 7.25) Householder speed with varying number of buses

represented as part of the network or a change in the network topology .

7.2.4 COMMUNICATIONS TIMING

The time taken in communicating between processors is the main bottleneck in many parallel processor systems . As more processors are added to the system, more time is consumed in keeping them all supplied with the necessary data . The M68000 systems used an IEEE bus to communicate which allowed each processor to send data to any number of other processors under control of the master processor . For the M68020 system a bus has been designed, along the same lines as the IEEE bus, using a 32 bit data bus instead of an 8 bit one . Timings were taken for the IEEE bus and then modified to represent the performance available from the new bus . Two types of data transfer are used by the simulator :

First, any processor can send data to one or all of the other processors . This type of transfer occurs in the initial set up and when the Householder routine runs because of a change in the network data .

Secondly, each slave in turn broadcasts to all the other processors along the bus . This happens during each pass through the generator routine and at the end of the network solution .

The first type of transfer is the simplest and the time spent in the transfer is:

transfer to one processor :

$$T_{c1}(\text{sec}) \simeq \frac{nw + 1}{1000000} \qquad (7.2.6)$$

transfer to all processors :

$$T_{c2}(\text{sec}) \simeq \frac{nw + ns}{1000000} \qquad (7.2.7)$$

where nw is the number of 32 bit words to be transferred and ns is the number of slave processors in the multiprocessor system . For the second type of transfer, if each slave has to broadcast nw words along the bus to all other processors the total time is :

$$T_{c3}(\text{sec}) \simeq \frac{ns*(nw+2)-2}{1000000} \qquad (7.2.8)$$

Because the new bus is controlled by a direct memory access controller, the processor can continue working while the data is being input or output . However, during the broadcasts the processor has to wait for all the new data before starting the new section of code .

7.3 OVERALL TIMING

Using equations (7.2.1) to (7.2.8) the total time for simulating any size of network on any number of processors can be calculated .

If the number of buses in the network is nb, the number of loads not represented by constant impedances is nl, the number of generators is ng and the number of slave processors in the multiprocessor is ns then the total computation time for a single time step can be estimated by the following method :

$$nga = INT \left(\frac{ng}{ns} + 0.5 \right) \qquad (7.3.1)$$

$$nla = INT \left(\frac{nl}{ns} + 0.5 \right) \qquad (7.3.2)$$

$$nba = INT \left(\frac{nb}{ns} + 0.5 \right) \qquad (7.3.3)$$

$$ni = ng_1 + nl_1 \qquad (7.3.4)$$

Equations (7.3.1) to (7.3.3) give the largest number of generators, buses and loads that a processor will have to deal with when the network is divided equally between the slave processors . Equation (7.3.4) gives the number of current injections in the system: for this equation nl is modified to be the number of loads represented which do not occur on generator buses (nl_1) . The number of generators is also modified to the number of buses which contain generators (ng_1) .These modifications are made because the injections at any bus can be summed to give a single injection to save time in the network solution .

Using these values, the time for each part of the algorithm can be calculated . These values are then used in the calculation of equations (7.2.1) to (7.2.5) . The resultant times are then combined to give an overall calculation time . The communication time is also dependent on

ng,nb and ns and can be calculated using equations (7.2.6) to (7.2.8) . The timing can be split into two parts, the steady state time for solution and the time required to perform a network change .

7.3.1 STEADY STATE SOLUTION TIME

The steady state solution time is the time taken by the simulator if no changes are made to the network: both the computation time and communication time are included . The time is given by :

$$T_{ss} = T_g + T_l + 2*T_n + 2*T_{com}^{1} + 2*T_{com}^{2} \qquad (7.3.5)$$

where T_g,T_l and T_n are calculated from equations (7.2.1), (7.2.2) and (7.2.4) respectively . T_{com}^{1} is the result of equation (7.2.8) to represent the data transfer at the end of the network solution which is carried out twice . The number of 32 bit words transferred by each processor (nw) is equal to nba*2 (all bus voltages are broadcast, each being a complex quantity occupying 2 words of storage). T_{com}^{2} is again the result of equation (7.2.8), this time to represent the transfer at the end of each pass through the generator and load routines . The number of 32 bit words transferred for this broadcast is ngl*2 .

7.3.2 NETWORK CHANGE TIME

The time taken for any change in the system depends on the type of change performed . A change in the generator set points (or load level for a load modelled separately from the network) is only a matter of updating two variables in the particular slave processor dealing with that generator (or load) . This means that the master processor need only send the type of change required, the generator number affected and the new power and frequency set points to one slave . The timing is obtained from equation (7.2.6) with nw having a value of four .

$$T_{nc1} = 5.0*10^{-6} \qquad (7.3.6)$$

The time taken for a pass through the Householder routine is governed by equation (7.2.8) and the data transfer part way through the calculation . The data transferred is an entire column of the inverted matrix and a complex scalar: this amounts to (2*nb + 2) words to be transferred . Thus the timing for this type of network change is :

$$T_{nc2} = T_h + T_{c2}' \qquad (7.3.7)$$

where T_{c2}' is the value of equation (7.2.7) using (2*nb + 2) as nw . If the change is a line outage or replacement then two passes through the routine are needed . If the change is to one of the loads modelled as part of the network then only one pass is required .

The total solution time is a combination of the times calculated from equation 7.3.5 and 7.3.7 . Because the time taken is different depending on whether or not the Householder routine is run there are two time steps for use in the simulator :

$$\Delta t_1 = T_{sm} + X*T_{nc1} \qquad (7.3.8)$$

and

$$\Delta t_2 = \Delta t_1 + Y*2*T_{nc2} \qquad (7.3.9)$$

Δt_1 gives the time step used when no network changes are required: it allows time for the alteration of a series of generator set points or load changes if the loads are modelled separately from the network . Δt_2 is the time step used when the network is changed and allows time for a number of passes through the Householder routine for the removal or replacement of lines or the alteration of constant impedance loads .

The choice of the values for X and Y depends on the size of the system and the length of T_{sm} . If T_{sm} is large then X and Y must be made large enough to cope with as many changes as are expected to occur in that time step . If more changes occur than can be performed in the time step, some are performed while the rest are performed before the next time step .

7.4 SIMULATION TIMING ESTIMATES

A series of estimates was calculated for the values of Δt_1 and Δt_2, using the method described above, for different sizes of system and numbers of slave processors . These estimates are listed in table 7.2 overleaf . The form of network routine used for the estimates is the injected current form so that the loads can be represented separately from the network .

Two sets of estimates were made . The first set compares the speed of execution of the method using all the loads as constant impedances with the method using the injected current loads . The second set of estimates examines the effect of different numbers of processors on the largest network .

7.4.1 LOAD MODEL COMPARISON TIMINGS

The timings for the two types of load model were estimated for five networks . Only the two extremes were examined . It is possible to model some of the loads as constant impedance and some as injected current, which gives a timing between the two presented in table 7.2 .

The timings show that the difference in timing between the two methods depends upon the number of loads which are not on their own buses (nll) . If this is large compared to the number of generators then the time increase for the second method is large .

nb	ng	nl	ns	nll	ngl	X	Y	T_{ss}	T_m	Δt_1	Δt_2
5	2	0	1	0	2	1	1	1.0446	0.96874	1.0496	3.01307
5	2	4	1	3	2	1	1	1.7796	0.96874	1.7846	3.74807
30	6	0	1	0	6	1	1	8.406	20.3223	8.411	49.1816
30	6	21	1	19	6	1	1	29.4635	30.3223	29.4685	70.2391
60	20	0	3	0	18	5	1	15.3858	28.8665	15.4108	73.3919
60	20	39	3	32	18	5	1	38.9946	28.8665	39.0197	97.0007
150	117	0	10	0	102	10	1	60.2692	57.4772	60.3192	175.898
150	117	111	10	33	102	10	1	79.1380	57.4772	79.1880	194.766
400	300	0	20	0	227	10	2	170.697	192.059	170.747	942.273
400	300	300	20	66	227	10	2	219.890	192.059	219.940	991.466

Table 7.2 Timing estimates for various networks (milliseconds)

7.4.2 MULTIPROCESSOR SIZE COMPARISON TIMINGS

The timing estimates for the 400 bus network above were repeated with a varying number of slave processors in the multiprocessor . This was done for both types of load model and the results are shown in figures 7.26 and 7.27: these results are also presented in tabular form in Appendix 3 .

Figure 7.26 shows that the general trend is for an increase in the number of processors to decrease the time taken in performing the simulation . As the number of processors increases the amount of time spent in communication between the processors increases, so the decrease is not linear . The portions of the graph where the time

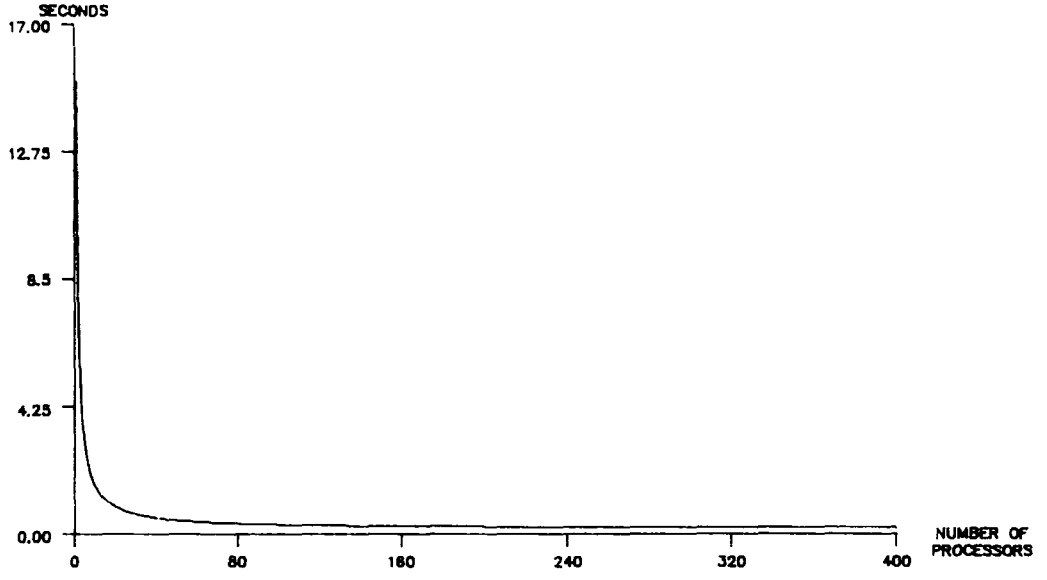


Figure 7.26) Δt_s using constant impedance loads

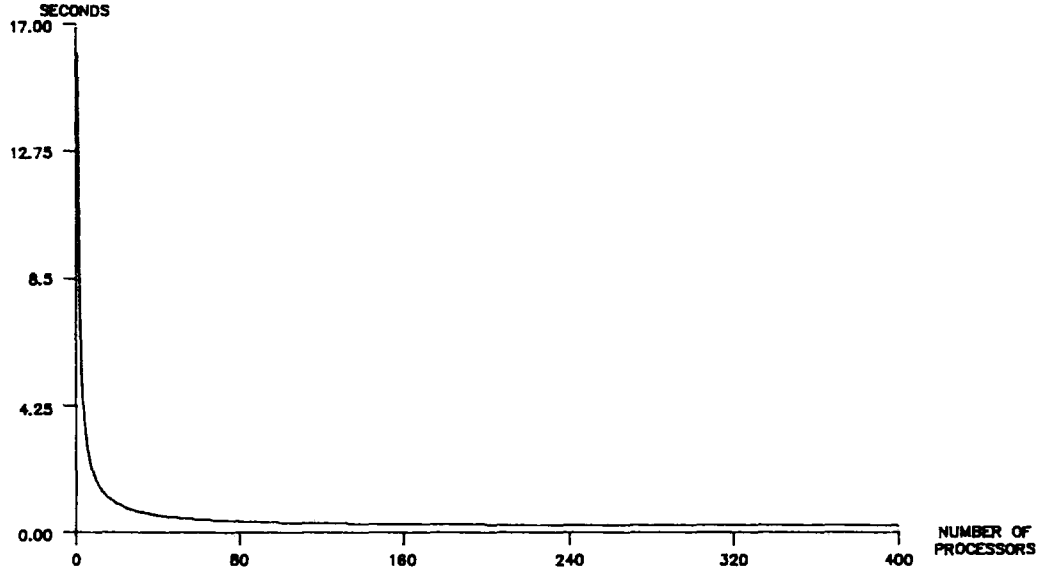


Figure 7.27) Δt_s using general load model

increases when a processor is added are due to the number of buses, generators and loads calculated by any one area (nba,nga,nla) remaining the same . This means the communication time rises but the solution time remains the same .

7.5 SIMULATOR HARDWARE REQUIREMENTS

The memory requirements for data for each routine can be calculated for any size of network and any number of processors . The size of the program code for the slave processors is very small, less than 20 kilobytes, and is constant regardless of the size of the network or number of slaves used . The master processor code is even smaller than the slaves, less than 10 k .

The processors all have at least half a megabyte of memory which would enable a single processor to run the largest of the systems given in table 7.2 . Even if more memory is required the processors are easily expandable and with a 32 bit address bus the theoretical memory limit is 4 gigabytes (2^{32}bytes) .

The use of the floating point chips was also investigated . The cost of these chips nearly doubles the price of each processor unit and this cost could not be justified if cheaper, less powerful chips could perform the same function . A logic analyser was connected to the floating point board to see how often it was in use . During the computational parts of both the network and Householder routines the board was in use approximately 80% of the time . The generator and load routines both

showed over 70% usage of the board . Thus any decrease in the speed of the floating point operations , through the use of different hardware, would seriously affect the performance of the simulator .

8 CONCLUSIONS

This thesis has covered the development of both the software and the hardware for a new method for simulating the behaviour of electrical power systems using multiple processors . Unlike much of the work in the field of dispatcher training simulators the design is intended to be used on a number of microprocessor based boards rather than two or three mini computers . This means that the system is expandable as the size of the power system to be simulated increases and can therefore continue to produce accurate, real time results more easily .

The advantages of using multiple processors for computing tasks have been utilised for some time in several fields . However only recently have technological advances made it economically viable to connect several processors in parallel for solving the complex sets of equations involved in power system simulation . Research by D.M. Detig [16] highlights four main problems with the use of special hardware for power system simulation which have to be overcome by any systems produced :

1) The form of the hardware often does not fit easily into the solution method used, especially since most of the algorithms used are simply modifications of the standard sequential algorithms in use on standard computers .

2) There is a lack of small, computationally intensive routines which are the type best suited to running in terms of the increase in

speed of execution obtainable by switching from serial to parallel computing .

3) The communication between the processors is very expensive in terms of time used but is an essential part of any parallel processing system .

4) The algorithms used in modelling various parts of the power system are constantly being revised and this could cause large changes in parallel systems because of the relative difficulty of developing code for them .

Finally she suggested that the amount of work required to overcome these problems would provide only a disproportionally small increase in speed . However with the increase in capability and decrease in cost of microprocessor hardware in the last five years this position has changed dramatically.

The use of a ready-designed multiprocessor, such as the CM* system, was ruled out and the research was carried out in such a way that the software and hardware designs evolved together to produce an overall solution to the simulation problem . This method meant that the inefficiency of forcing an algorithm onto a piece of hardware to which it is not ideally suited is largely avoided . The algorithm developed is radically different from those used on single processor systems . Also, unlike some of the parallel algorithms, the new method requires no central

computational stage, where all but one of the processors lie idle, except for one part of the calculation if a line is outaged.

The program size of the computational loops of the generator,load and network routines, when coded in C using the floating point hardware, were all less than 256 bytes and therefore fitted into the microprocessors internal high speed cache memory . Using this memory increased the execution speed of the inner loops by approximately 25% . Within these loops the floating point hardware was being utilised between 75% and 80% of the time . Thus the main parts of the simulator were in a form that was well suited to the hardware being used and utilised the available resources to the full . Using a slower (but cheaper) floating point processor such as the Motorolla M68881 would have a significant affect on the overall performance of the simulator . The algorithm as a whole did not use a large amount of pointers and data movement, as are required by sparse matrix techniques, which cannot easily be economically split onto multiprocessor systems .

Keeping the amount of communication between the processors low was one of the prime objectives of the original software design specification and the final algorithm required little inter-processor communication . The communication between the host computer and the simulator is fixed depending upon the type of simulator required . For example a training simulator should produce the same amount of data as the telemetry system on the real power system . The master processor which coordinates the communication between the other processors and also communicates with the host does not perform any of the simulation tasks . Even the data corruption which the master does perform could be transferred to the host

computer if the speed became critical . The use of a communication bus rather than shared memory also allows the master processor to control the synchronisation of the processors very easily . This synchronisation is simpler to achieve on a linear multiprocessor like the one used than on a grid type multiprocessor .

The last of the points raised by D.M. Detig was solved by separating the dynamic and static solutions, thus allowing the load and generation sections to be coded separately from the network model . Thus the generator or load model can be changed to reflect the latest requirements and, providing the input and output requirements of the routines are met, the new routines can simply be used in place of the existing ones . The network routine could not be changed so easily but even the facility of changing the dynamic models and being able to compare different solution techniques and models is an advantage over those methods which combine the dynamic and static portions of the simulation and require large program changes to implement new models .

Two test power systems were used to compare the results obtained by the developed multiprocessor algorithm and a standard sequential algorithm on a minicomputer . These test systems were the standard IEEE 5 and 30 bus systems, the results show a good correlation between the steady state results produced while the parallel algorithm modelled far more of the transient effects of changes in the power system . This is due to the faster execution of the multiprocessor simulator and hence the possibility of using a far shorter time step for the simulation . By reducing the sequential simulator's time step the correlation between the transients was also found to be very good . The software developed for the multiprocessor

was also more robust when events such as islanding occur . The sequential system used for comparison could, at best, only continue simulating the results in one island while the multiprocessor continued simulating the entire system even when several islands were formed and then brought together .

Comparing the algorithm to many of those produced to run a decomposed algorithm on two or three minicomputers shows two main advantages:

Firstly the major bottleneck caused by large amounts of centralised computation normally required by the decomposed method is avoided by the new algorithm . The only portion of central computation required being one small portion of the matrix alteration routine .

Secondly there are no rules to be followed over the dividing up of the system between processors . Many diakoptic methods restrict what types of buses can be connected to tie lines and nearly all work best when there are a minimum number of tie lines . The only guidelines for dividing the network with the new algorithm is to keep the simulation of the generators and loads on the same slave processor as that which simulates the bus to which they are connected . This does not cause any further work in trying to divide the system because it is natural to keep all of the functions of one bus on a single processor .

The estimated simulation times produced from the results obtained suggest that a realistic operator training simulator for a system of the size of the C.E.G.B.'s, comprising 90 generators and 150 nodes, could easily be solved on a multiprocessor . By using a simulator with 15 slaves a time step of 150 milliseconds could be achieved even if a matrix change had to be performed . With no matrix changes required the system could run with a time step of less than 50 milliseconds . Even with this number of processors the cost of building the multiprocessor hardware for such a system would still be less than the cost of a standard mini-computer, around £80,000 including the floating point hardware and communication interfaces .

The algorithm is also suitable for any type of use . By relaxing the need for real time simulation and including higher order generator and load models the simulator can, by using a smaller time step, be used to simulate the transients in the system accurately . The man-machine interfaces required for any of the uses to which the simulator can be put would all be resident on the host computer rather than on the multiprocessor . Therefore they can use the full facilities, such as graphics and mass storage media, available on the host through a high level language .

With devices purpose-built for parallel computation such as Transputers becoming more readily available the task of programming a multiprocessor system is greatly simplified . By using Occam, programs can be written in a high level language and have the interprocess communication rigidly defined . However the algorithm developed would not fit as readily onto a device such as the Transputer, which uses a grid

configuration, as onto the developed hardware which uses a linear configuration . This is due to the fact that the bus connecting the processors in the developed hardware allows broadcasts to all processors simultaneously . On a transputer system this can only be achieved by passing the data through the grid: however the algorithm could still be run on a transputer system successfully .

The new power system simulator described in this thesis makes use of some of the latest computer technology . Its major advantages over standard sequential simulators are; it is fast in producing results, flexible in both the systems which can be simulated and the uses to which the simulation can be put and finally economical in terms of hardware costs .

9 REFERENCES AND BIBLIOGRAPHY

9.1 REFERENCES

The following are books and papers which are directly referenced from within the text of the thesis:

1. **M. Rafian, M.J.H. Sterling and M.R. Irving,** "Real time power system simulation", Research report, Engineering department, University of Durham, 1986.

2. **R. Joetten, T. Weß, J. Wolters, H. Ring and B. Bjoernsson,** "A new real-time simulator for power system studies", IEEE Transactions PAS-104, No. 9, pp. 2604-2611, September 1985.

3. **R. Podmore, J.C. Giri, M.P. Gorenberg, J.P. Britton and M.M. Peterson,** "An advanced dispatcher training simulator", IEEE Transactions PAS-101, No. 1, pp.17-25, January 1982.

4. **K. Saikawa, M. Goto, Y. Imamura, M. Takato and T. Kanke,** "Real time simulation system of large-scale power system dynamics for a dispatcher training simulator", IEEE Transactions PAS-103, No. 12, pp. 3496-3501, December 1984.

5. **J.R. Latimer and R.D. Masiello,** "Design of a dispatcher training system", IEEE PICA Conference proceedings, pp. 87-92, 1977.

6. **H. Biglari, E.E. Cashar, K. Hemmaplardh, D.K. Lee, H. Ramchandani and S.A. Sackett,** "A dispatcher training simulator design with multi purpose interfaces", IEEE Transactions PAS-104, No. 6, pp. 1276-1280, June 1985.

7. **D. Magee, F. Flynn, P. Wehlage, J. Waight and R.K. Lehman,** "A large two computer dispatcher training simulator", IEEE Transactions PAS-104, No. 6, pp. 1433-1438, June 1985.

8. **I. Susumago, M. Suzuki, K. Miyama, T. Tsuji, K. Dan and A. Yamanishi,** "Development of a large scale dispatcher training simulator", IEEE Transactions PWRS-1, No. 2, pp. 67-75, May 1986.

9. **K. Sato, Z. Yamazaki, T. Haba, M. Fukushima, K. Masegi and H. Hayashi,** "Dynamic simulation of a power system network for dispatcher training", IEEE Transactions PAS-101, No. 10, pp. 3742-3750, October 1982.

10. **G.R. Ott, L.M.Walker and D.T.Y. Wong,** "Hybrid simulation for long term dynamics", IEEE Transactions PAS-96, No. 3, pp. 907-915, May/June 1977.

11. **A. Keyhani,** "Development of an interactive power system research simulator", IEEE Transactions PAS-103, No. 3, pp. 516-521, March 1984.

12. **B. Stott and O. Alsac,** "Fast decoupled load flow", IEEE Transactions PAS-93, pp. 859-869, 1974.

13. **W.F. Tinney and C.E. Hart,** "Power flow solution by Newton's method", IEEE Transactions PAS-86, pp. 1449-1460, November 1974.

14. **L. Elder and M.J. Metcalfe,** "An efficient method for real time simulation of large power system disturbances", IEEE Transactions PAS-101, No. 2, pp. 334-339, February 1982.

15. **H.H. Happ, C. Pottle and K.A. Wirgau,** "Future computer technology for large power system simulation", IFAC, Vol. 15, pp. 621-629, August 1979.

16. **D.M. Detig,** "Effects of special purpose hardware in scientific computation with emphasis on power system applications", IEEE Transactions PAS-101, No. 2, pp. 265-270, February 1982.

17. **M. Takatoo, S. Abe, T. Bando, K. Hirasawa, M. Goto, T. Kato and T. Kanke,** "Floating vector processor for power system simulation", IEEE Transactions PAS-104, No. 12, pp. 3361-3366, December 1985.

18. **H.H. Happ,** "Parallel processing in power systems", 7th PSCC conference, Lausanne, 1981.

19. J. **Fong and** C. **Pottle,** "Parallel processing of power system analysis problems via simple parallel microcomputer structures", IEEE Transactions PAS-97, No. 6, pp. 1834-1840, September/October 1978.

20. **F.M. Brasch, J.E. Van Ness and** S.C. **Kang,** "Simulation of a multiprocessor network for power system problems", IEEE Transactions PAS-101, No. 2, pp. 295-301, February 1982.

21. **W.L. Hatcher, F.M. Brasch and** J.E. **Van Ness,** "A feasibility study for the solution of transient stability problems by multiprocessor structures", IEEE Transactions PAS-96, No. 6, pp. 1789-1797, November/December 1977.

22. **R. Lopez-Lopez,** "Dynamic simulation of power systems on multiple microprocessors", Ph.D. Thesis, Electrical Engineering Department, Imperial College of Science and Technology, University of London, November 1983.

23. **S.M. Talukdar and D. Thomas,** "Modular algorithms and multi processors for simulating power systems", EPRI report EL-566-SR : Exploring applications of parallel processing to power systems, pp. 325-334.

24. **L. A. Dale, A. R. Daniels and I. A. Erinmez,** "The real-time modelling of the operation of complex power systems", Proceedings of the 21st universities power engineering conference, pp. 181-183, 1986.

25. **H. Mukai,** "Parallel algorithms for solving systems of nonlinear equations", Computing and maths with applications, Vol 7, pp. 235-250, Pergamon press, 1981.

26. **S.H. Fuller, J.K. Ousterhout, L. Raskin, P.I. Rubinfield, P.J. Sindhu and R.J. Swan,** "Multi-microprocessors: An overview and working example", Proceedings of the IEEE, Vol. 66, No. 2, pp. 216-228, February 1978.

27. **A.K. Jones, R.J. Chansler, I. Durham, P.H.Feiler, D.A. Scelza, K. Schwans and S.R. Vegdahl,** "Programming issues raised by a multiprocessor", Proceedings of the IEEE, Vol. 66, No. 2, pp. 229-237, February 1978.

28. **J. Grosser and S.M.Talukdar,** "Models for MIMD machines", IEEE Transactions PAS-101, No. 1, pp. 94-99, January 1982.

29. **A.J. Perry,** "A multiprocessor simulator for an electrical power system", third year project report, Engineering department, University of Durham, 1982.

30. **J.C.H. Thomson,** "A multiprocessor simulator for an electrical power system", third year project report, Engineering department, University of Durham, 1983.

31. **T.J. Hammons and D.J. Winning,** "Comparisons of synchronous machine models in the study of the transient behaviour of electrical power systems" Proceedings of the IEE, Vol. 118, No. 10, pp. 1442-1458, October 1971.

32. **S.R. Macminn and R.J. Thomas,** "Microprocessor simulation of synchronous machine dynamics in real time", IEEE Transactions PWRS-1, No. 3, pp. 220-225, August 1986.

33. **P.L. Dandeno, R.L. Hauth and R.P. Schulz,** "Effects of synchronous machine modelling in large scale system studies", IEEE power engineering society 1972 summer meeting.

34. **G. Shackshaft, O.C. Symons and J.G. Hardwick,** "General purpose model of power system loads", Proceedings of the IEE, Vol. 124, No. 8, pp. 715-723, August 1977.

35. **C. Concordia and S. Ihara,** "Load representation in power system stability studies", IEEE Transactions PAS-101, No. 4, pp. 969-977, April 1982.

36. **M.H. Kent, W.R.Schmus, F.A. McCrackin, and L.M. Wheeler,** "Dynamic modelling of loads in stability studies", IEEE Transactions PAS-88, No. 5, pp. 756-763, May 1969.

37. **M. Langevin and P. Auriol,** "Load response to voltage variations and dynamic stability", IEEE Transactions PWRS-1, No. 4, pp. 112-118, November 1986.

38. **K. Zollenkopf,** "Bi-factorisation - basic computational algorithm and programming techniques", Large sparse sets of linear equations, J.K. Reid (Ed.), Academic press, 1971.

39. **H.H. Happ**, "Piecewise methods and applications to power systems", John Wiley & Sons, New York, 1980.

40. **M. El-Marsafawy, R.W. Menzies and R.M. Mathur**, "A new, exact, diakoptic, fast-decoupled load-flow technique for very large power systems", IEEE power engineering society 1979 summer meeting.

41. **A.S. Householder**, "Principles of numerical analysis", McGraw-Hill, New York, 1953.

42. **W.G. Stagg and A.H. El-Abiad**, "Computer methods in power system analysis, McGraw-Hill, New York, 1968.

43. **I. Durham, R.C. Dugan and S.M. Talukdar**, "An algorithm for power system simulation by parallel processing", IEEE power engineering society 1979 summer meeting.

44. **I. Durham, R.C. Dugan, A.K. Jones and S.M. Talukdar**, "Power system simulation on a multiprocessor", IEEE power engineering society 1979 summer meeting.

45. **C. Pottle**, "Solution of sparse linear equations arising from power system simulation on vector and parallel processors", Joint automatic control conference, 1978.

46. **L.L. Ferris and A.M. Sasson**, "Investigation of the load flow problem", Proceedings of the IEE, Vol. 115, No. 10, pp. 1459-1469, October 1968.

9.2 BIBLIOGRAPHY

The following are books and papers which, though not referenced in this thesis, are relevant to certain areas of the research:

A. Brameller, M.N. John and M.R. Scott, "Practical diakoptics for electrical networks", Chapman and Hall Ltd, 1969.

P.L. Dandeno and P. Kundur, "A non iterative transient stability program including the effects of variable load voltage characteristics", IEEE power engineering society 1973 winter meeting.

P.M. Dew, T.F. Buckley and M. Berzins, "Application of VLSI devices to computational problems in the gas industry", Research report, Computing department, University of Leeds, 1982.

Erisman, Neves and Dwarakanath, "Electrical power problems : The mathematical challenge", A collection of papers for the Society for Industrial and Applied Mathematics (SIAM) conference, Philadelphia, 1980.

C.A. Gross, "Power system analysis", John Wiley & Sons, New York, 1979.

G.D. Hatchel and A.L. Sangiovanni-Vincentelli, "A survey of third-generation simulation techniques", Proceedings of the IEEE, Vol. 69, No. 10, pp. 1264-1280, October 1981.

J.M. Ortega and W.C. Rheinboldt, "Iterative solutions of nonlinear equations in several variables", Academic Press, 1970.

M. Rafian, M.J.H. Sterling and M.R. Irving, "Decomposed load flow", Research report, Engineering department, University of Durham, 1984.

A. Ralston and H.S. Wilf, "Mathematical methods for digital computers", John Wiley & Sons, New York, 1960.

A.P. Sage and S.L. Smith, "Real-time digital simulation for systems control", Proceedings of the IEEE, Vol. 54, No. 12, pp. 1802-1812, December 1966.

B. Stott, "Decoupled Newton load flow", IEEE Transactions PAS-91, pp. 1955-1959, Sept/Oct 1972.

R. Wait, "The numerical solution of algebraic equations.", John Wiley & Sons, New York, 1979.

Y. Wallach, "Alternating sequential / parallel processing", Springer-Verlag, New York, 1982.

"Power Engineering Review", IEEE monthly periodical.

APPENDIX 1

DIAKOPTIC CUT LINE MATRIX

The method discussed in section 3.3 of this thesis splits the B' and B" matrices into three parts . To reduce the memory requirements for the algorithm the matrices representing the inter area lines (B'_{iml} and B''_{iml}) can both be represented by a connection matrix and a square matrix containing the elements of the corresponding B matrix . Looking at the B' matrix :

$$B'_{iml} = CM'C^{t} \qquad (10.1.1)$$

where C is the connection matrix and M' the element matrix . The C matrix is rectangular with the same number of rows as B'_{iml} while the number of columns is equal to the number of cut lines . All the elements C_{ij} are either 1 (first end of cut line j is at bus i), 0 (cut line j does not touch bus i) or -1 (second end of cut line j is at bus i) . The M' matrix is square with the number of rows equal to the number of cut lines: the off diagonal elements are all zero while the diagonal element M'_{jj} contains the 1/x value for line j .

This process can be repeated with the B" matrix: the C matrix is exactly the same so does not need to be recalculated and is only stored once . The M" matrix contains the negated susceptance terms for the tie lines on the diagonal and the rest of the terms are zero .

The B'_{ctb} and B''_{ctb} matrices are also very sparse and can be held as rectangular matrices because only the columns and rows corresponding to the temporary buses have any elements: since the matrices are symmetrical only the column half need be stored and the transpose of this used for the row section .

APPENDIX 2

TEST SYSTEM PARAMETERS

The parameters for the 30 bus system (figure 7.1) are slightly different from those given for the standard IEEE test network as presented by L.L. Ferris and A.M. Sasson [46] . They include typical generator time constants and inertias and the load data used was taken from a midnight load distribution on a theoretical load curve .

The system contains 30 buses, 6 generators, 41 lines and 21 loads . The data, on a per unit base of 100 MW is as follows :

GENERATOR PARAMETERS

GENERATOR NUMBER	1	2	3	4	5	6
BUS NUMBER	1	2	5	8	11	13
POWER	0.50539 + j 0.28684	0.50000 + j 0.18027	0.30339 + j 0.12843	0.50000 - j 0.05873	0.50000 + j 0.28684	0.10000 + j 0.29043
TRANS REACT	0.00000 + j 0.25000	0.00000 + j 0.25000	0.00000 + j 0.50000	0.00000 + j 0.50000	0.00000 + j 0.50000	0.00000 + j 0.50000
GAIN	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
H	5.00000	5.00000	4.00000	3.00000	4.00000	4.00000
TC	0.30000	0.30000	0.30000	0.30000	0.30000	0.30000
PSET	0.50539	0.50000	0.30339	0.50000	0.50000	0.10000
FSET	50.0000	50.0000	50.0000	50.0000	50.0000	50.0000

LINE PARAMETERS

LINE NUMBER	BUSES CONNECTED	RESISTANCE P.U.	REACTANCE P.U.	LINE CHARGING P.U.
1	1 - 2	0.0192	0.0575	0.0264
2	1 - 3	0.0452	0.1852	0.0204
3	2 - 4	0.0570	0.1737	0.0184
4	3 - 4	0.0132	0.0379	0.0042
5	2 - 5	0.0472	0.1983	0.0209
6	2 - 6	0.0581	0.1763	0.0187
7	4 - 6	0.0119	0.0414	0.0045
8	5 - 7	0.0460	0.1160	0.0102
9	6 - 7	0.0267	0.0820	0.0085
10	6 - 8	0.0120	0.0420	0.0045
11	6 - 9	0.0000	0.2080	0.0000
12	6 - 10	0.0000	0.5560	0.0000
13	9 - 11	0.0000	0.2080	0.0000
14	9 - 10	0.0000	0.1100	0.0000
15	4 - 12	0.0000	0.2560	0.0000
16	12 - 13	0.0000	0.1400	0.0000
17	12 - 14	0.1231	0.2559	0.0000
18	12 - 15	0.0662	0.1304	0.0000
19	12 - 16	0.0945	0.1987	0.0000
20	14 - 15	0.2210	0.1997	0.0000
21	16 - 17	0.0824	0.1923	0.0000
22	15 - 18	0.1070	0.2185	0.0000
23	18 - 19	0.0639	0.1292	0.0000
24	19 - 20	0.0340	0.0680	0.0000
25	10 - 20	0.0936	0.2090	0.0000
26	10 - 17	0.0324	0.0845	0.0000
27	10 - 21	0.0348	0.0749	0.0000
28	10 - 22	0.0727	0.1499	0.0000
29	21 - 22	0.0116	0.0236	0.0000
30	15 - 23	0.1000	0.2020	0.0000
31	22 - 24	0.1150	0.1790	0.0000
32	23 - 24	0.1320	0.2700	0.0000
33	24 - 25	0.1885	0.3292	0.0000
34	25 - 26	0.2544	0.3800	0.0000
35	25 - 27	0.1093	0.2087	0.0000
36	27 - 28	0.0000	0.3960	0.0000
37	27 - 29	0.2198	0.4153	0.0000
38	27 - 30	0.3202	0.6027	0.0000
39	29 - 30	0.2399	0.4533	0.0000
40	8 - 28	0.0636	0.2000	0.0214
41	6 - 28	0.0169	0.0599	0.0065

The line charging given in the final column is half the total charging for the line .

LOAD DATA

LOAD NUMBER	BUS NUMBER	ACTIVE POWER	REACTIVE POWER
1	2	0.18221	0.10664
2	3	0.02015	0.01008
3	4	0.06381	0.01344
4	5	0.79096	0.15953
5	7	0.19144	0.09152
6	8	0.25190	0.25190
7	10	0.04870	0.01697
8	12	0.09404	0.06297
9	14	0.05206	0.01344
10	15	0.06885	0.02099
11	16	0.02939	0.01511
12	17	0.07557	0.04870
13	18	0.02687	0.00756
14	19	0.07977	0.02855
15	20	0.01847	0.00588
16	21	0.14694	0.09404
17	23	0.02687	0.01344
18	24	0.07305	0.05626
19	26	0.02939	0.01931
20	29	0.02015	0.00756
21	30	0.08900	0.01595

APPENDIX 3

TIMING RESULTS

The results for the tests giving the time taken in the network and Householder routines are presented in graphical form in chapter 7 . They are given here in tabular form :

NETWORK TIMING

Number of milliseconds for a network solution with varying numbers of generators (ng) and buses in each area (nba) .

NUMBER OF BUSES	NUMBER OF GENERATORS					
	2	6	15	30	60	120
5	0.3646	1.0216	2.542	5.048	10.06	19.92
10	0.6704	1.9065	4.743	9.486	18.88	37.48
15	0.9980	2.8300	7.000	13.93	27.72	54.74
20	1.2890	3.725	9.207	18.31	37.63	72.10
30	1.8880	5.500	13.61	27.32	53.96	107.2
50	3.134	9.035	22.36	44.36	88.72	176.0
75	4.667	13.59	33.36	66.44	132.3	263.1
100	6.230	17.99	44.33	88.30	176.4	349.9
128	7.850	22.84	56.77	113.3	226.0	449.6

All these results were used to obtain the series of straight line fits which are presented in the results section on the graphs and as equation (7.2.1) .

HOUSEHOLDER TIMINGS

Number of milliseconds for a Householder inverted matrix alteration for varying numbers of buses in the system (nb) and buses in each area (nba) .

NUMBER OF BUSES IN AREA	NUMBER OF BUSES IN SYSTEM								
	5	10	15	20	30	50	75	100	128
5	1.011	--	--	--	--	--	--	--	--
10	1.508	2.939	--	--	--	--	--	--	--
15	2.010	3.932	5.846	--	--	--	--	--	--
20	2.507	4.939	7.304	9.737	--	--	--	--	--
30	3.504	6.896	10.282	13.696	20.212	--	--	--	--
50	5.447	10.867	16.260	21.573	31.920	53.200	--	--	--
75	7.980	15.820	23.475	31.080	46.500	77.600	116.29	--	--
100	10.467	20.667	31.080	41.059	61.360	102.00	153.00	203.50	--
128	13.250	26.400	39.400	51.733	77.700	129.50	194.50	258.67	330.40

All these results were used to obtain the series of straight line fits which are presented in the results section, both on the graphs and as equation (7.2.3) .

APPENDIX 4

CODE EXAMPLES

The first listing is the assembler listing for the master processor in the M68000 multiprocessor system, it does not include the listing for the floating point software or the IEEE interface drivers . The generator algorithm is in two parts:

1) The initialisation of the generator variables which is the section after the label "genint:" .

2) The main simulation algorithm which is the section after the label "gen:" .

Similarly the network simulation is performed in two parts, the initialisation after the label "netint:" and the main simulation section after the label "net:".

The second, third and fourth listings are the 'C' programs written for the M68020 system with the floating point hardware . These all use macro definitions to access the floating point hardware, these macros (FFPU and IFPU) take three arguments . The first argument specifies whether to use the multiplier chip or the arithmetic chip:

SEQ4 - means put a parameter into the arithmetic chip .

SEQ5 - means get an answer from the arithmetic chip .

SEQ6 - means put a parameter into the multiplier chip .

SEQ7 - means get an answer from the multiplier chip .

The second argument specifies which register in the floating point chip to load or unload:

LAS - load the A side of the appropriate chip .

LBSF - load the B side of the appropriate chip and perform a function .

UNLOAD - unload the answer register of the appropriate chip .

The final argument specifies which of the possible functions the arithmetic unit is to perform (the multiplier will always perform a multiply):

SA - single precision add .

SS - single precision subtract .

SD - single precision divide .

The second listing gives the algorithm used for the network calculations, the third gives the generator algorithm and the final listing gives the householder matrix alteration program . All these listings are those used for timing calculations and do not include any interprocessor communication.

```
        .data
;       strings for output

stino:  .asciz  "SIMCOM\r"

next:   .asciz  "\r\nnext:-"

gstrop: .asciz  "\r\n set points for generator numbe: "

astrop: .asciz  " altered \r\n"

lstrop: .asciz  "\r\n value of load number "

tstrop: .asciz  "\r\n line number "

ostrop: .asciz  " outaged\r\n"

istrop: .asciz  " put back in\r\n"

asknsl: .asciz  " input number of slaves in hex \r\n"

genstl: .asciz  "\r\n\r\n generater initialise about to start \r\n"
netstl: .asciz  "\r\n network initialise about to start \r\n"
stalst: .asciz  "\r\n back from stalls"
sta2st: .asciz  "\r\n go to stocom"
inopst: .asciz  "\r\n input line number please \r\n"
linotp: .asciz  "\r\n in or out (1 or 2) \r\n "

        .even

altno:  .long   0x00000000
inout:  .long   0x00000000
        .even

; STORAGE FOR HOUSEHOLDER ROUTINES

temp:   .       =       3+.
temp1:  .       =       3+.
temp2:  .       =       3+.
buf:    .       =       30+.
nslave: .       =       4+.

;       MAIN PROGRAM
        .text
        .even
        .globl  _main
_main:  movl    #0,temp1
        movl    #asknsl,a0      ; ask for number of slaves in system
        trap    #15
        .word   1
        movl    #buf,a5
        trap    #15             ; read in buffer from keyboard
        .word   5
```

```
        trap    #15             | decode into register d0
        .word   12
        movl    d0,nslave
        jbsr    csetup          | initialise controler for case bus
        jbsr    datin           | get data from host

        movl    nslave,d0
        movl    #0x0,d5
        movl    ng,d1
        movl    #slgeno,a0
rediv:  movl    d1,d2
        divu    d0,d2
        movl    d2,d3
        andl    #0x0000ffff,d2
        andl    #0xffff0000,d3
        beq     noad1
        addl    #1,d2
noad1:  subl    d2,d1
lop1:   movl    d5,a0@+         | slgeno contains slave number for each gen
        subl    #1,d2
        jne     lop1
        addl    #1,d5
        cmpl    #0x0,d1
        bne     rediv


        movl    nslave,d0
        movl    nb,d1
        movl    #slbsno,a0
rediv2: movl    d1,d2
        divu    d0,d2
        movl    d2,d3
        andl    #0x0000ffff,d2
        andl    #0xffff0000,d3
        beq     noad2
        addl    #1,d2
noad2:  movl    d2,a0@+         | slbsno contains number of busses for slave
        subl    #1,d0
        subl    d2,d1
        bne     rediv2

        movl    #gensti,a2
        trap    #15
        .word   10
        jbsr    genint          | initialise generators
        movl    #netsti,a2
        trap    #15
        .word   10
        jbsr    netint          | initialise networks
        jbsr    volout

sec20:  movl    #10,temp2
seclp:
        jbsr    gen
        jbsr    net
        jbsr    start
```

```
                subl    #1,temp2
                bne     sec1b
                jsr     volout
                jra     sec20

volout:         movl    temp1,d0
                lsll    #3,d0
                addl    #busvol,d0
                movl    d0,a0
                jsr     outbeh
                jsr     outbeh
                movl    #0x0a,d0
                trap    #15
                .word   2
                movl    temp1,d0
                addl    #1,d0
                cmpl    #b,d0
                jne     noteq1
                movl    #0,d0
noteq1:         movl    d0,temp1
                rts

start:          trap    #15
                .word   5
                cmpl    #0,d0
                jne     around
                movl    #0xffffffff,iel
                jsr     ncont
                rts
around:         jsr     gtenum
                cmpb    #0,d0
                jeq     gchr            | goto generator change routine
                cmpb    #1,d0
                jeq     lochr           | goto load change routine
                cmpb    #2,d0
                jeq     lichr           | goto line change routine
                cmpb    #3,d0
                jeq     sistop          | goto simulator stop routine
                jra     start


gchr:           jsr     gtenum
                movl    d0,d3
                movl    d0,altno
                jsr     gtenum          | ask for new pset
                movl    d0,d4
                jsr     gtenum
                subl    #1,d3
                asll    #2,d3
                addl    #pset,d3
                movl    d3,a0
                movl    d4,a00          | store pset
                subl    #pset,d3
                addl    #fset,d3
                movl    d3,a0
                movl    d0,a00          | store fset
```

```
            subl    #fset,d2
            addl    #icontr,d3
            movl    d3,a0
            movl    #1,a0@
            movl    #gstrop,a3
            trap    #15
            .word   10
            movl    #altno,a0
            movl    #4,d1
            trap    #15
            .word   11
            movl    #astrop,a3
            trap    #15
            .word   10
            jra     start

locbr:
            jsr     gtenum          | find which load is to be altered
            movl    d0,d3
            movl    d0,altno
            jsr     gtenum
            movl    d0,temp
            jsr     gtenum
            movl    d0,temp+4
            subl    #1,d3
            movl    d3,iel
            asll    #3,d3
            addl    #loadadm,d3     | calculate difference to be sent to house
            movl    d3,a1
            movl    #temp,a0
            movl    #v1,a2
            jsr     consub
            movl    iel,d3
            asll    #2,d3           | get load bus
            addl    #ilocbu,d3
            movl    d3,a3
            movl    a3@,d3
            subl    #1,d3
            movl    d3,iel
            movl    a0@+,a1@+
            movl    a0@+,a1@+       | update load admittance
            movl    #0,ie2
            movl    #0,v2           | set up variables for householder
            movl    #0,v2+4
            jsr     ncont           | go to householder routine
            movl    #1strop,a3
            trap    #15
            .word   10
            movl    #altno,a0
            movl    #4,d1
            trap    #15
            .word   11
            movl    #astrop,a3
            trap    #15
            .word   10
            jra     start
```

```
lichr:
        movl    #lnopst,a3
        trap    #15
        .word   10
        jbsr    gtenum          | find which line is to be altered
        movl    d0,d3
        movl    d0,altno
        subl    #1,d3
        asll    #2,d3
        movl    d3,d4
        addl    #icon,d3        | find whether line is in or out
        movl    d3,a4
        movl    a4@,inout
        movl    #linotp,a3
        trap    #15
        .word   10
        jbsr    gtenum
        movl    d0,d3
okOorl: cmpl    a4@,d3
        jeq     start
        movl    d4,d5
        movl    d4,d6
        addl    #iend1,d5
        addl    #iend2,d6       | find busses at both ends of the line
        movl    d5,a0
        movl    d6,a1
        movl    a0@,d5
        subl    #1,d5
        movl    d5,ie1
        movl    a1@,d6          | store buses-1 as ie1 and ie2 for house
        subl    #1,d6
        movl    d6,ie2
        asll    #1,d4
        movl    d4,d5
        addl    #linadm,d4
        addl    #linerg,d5      | set up v1 and v2 for house from
        movl    d4,a0           | lin admittance and line charging
        movl    d5,a1
        movl    #v1,a2
        jbsr    comadd
        movl    a0@+,v2         | v1=linadm(I)+linerg(I)
        movl    a0@,v2+4        | v2=linadm(I)
        movl    d3,a4@
        jeq     puttne
        eorl    #0x80000000,v2
        eorl    #0x80000000,v2+4
        jbsr    ncont
        jra     nexths          | one of v1 and v2 must be negated
puttne: eorl    #0x80000000,v1  | depending on insertion or deletion
        eorl    #0x80000000,v1+4
        jbsr    ncont
nexths: movl    ie1,temp
        movl    ie2,ie1
        movl    temp,ie2        | second call to house with ie1 and ie2 swapped
        jbsr    ncont
```

```
        movl    #tstrop,a3
        trap    #15
        .word   10
        movl    #altno,a0
        movl    #4,d1
        trap    #15
        .word   11
        cmpl    #1,inout
        jeq     oopst
        movl    #istrop,a3
        trap    #15
        .word   10
        jra     start
oopst:  movl    #ostrop,a3
        trap    #15
        .word   10
        jra     start

sistoo: trap    #15
        .word   14


pe2iee: movl    d0,d1           | convert f.p. number in d0 to ieee form
        andl    #0x7f000000,d1
        subl    #0x20400000,d1
        asll    #2,d1

        movl    d0,d2
        andl    #0x00ffffff,d2  | d2 contains fractional part
        jeq     zero

topset: lsll    #1,d2
        subl    #0x01000000,d1
        btst    #24,d2          | normalise number
        jeq     topset
        andl    #0x00ffffff,d2
        orl     d1,d2
        roxll   #1,d0           | reassemble in d2
        roxrl   #1,d2
        rts
zero:   movl    #0,d2
        rts



iee2pe: movl    d0,d1           | convert ieee to p.e. format for f.p.
        jeq     zero
        andl    #0x7f800000,d1
        addl    #0x40800000,d1
        lsrl    #1,d1

        movl    d0,d2
        andl    #0x007fffff,d2
        orl     #0x00800000,d2  | d2 contains the fractional part
        addl    #0x00400000,d1
```

```
divby4: btst    #22,d1
        jne     normal
        btst    #23,d1
        jeq     okasmb          Inormalise the number
normal: lsrl    #1,d2
        addl    #0x00400000,d1
        jra     divby4

okasmb: orl     d1,d2
        asll    #1,d0
        jcc     retbe           ! reassemble into b.e. format
        bset    #31,d2
retbe:  rts

datin:  jbsr    stcpd
        movl    #nb,a4          ! input number of busses
        jbsr    nexin
        movl    #nl,a4          ! input number of lines
        jbsr    nexin
        movl    #nlb,a4         ! input number of loads
        jbsr    nexin
        movl    #ng,a4          ! input number of generators
        jbsr    nexin
        movl    #busvol,a4
        movl    nb,d5
initbs: movl    #0x3f800000,a4@+
        movl    #0x0,a4@+       ! initialise bus voltages
        subl    #1,d5
        jne     initbs
        movl    #iend1,a4
        movl    nl,d5           ! input both ends of each line
        jbsr    arryin
        movl    #iend2,a4
        jbsr    arryin
        movl    #linadm,a4      ! input line admittance array
        jbsr    arrycp
        movl    #lincrg,a4      ! input lincrg array
        jbsr    arrycp
        movl    #icon,a4
initic: movl    #1,a4@+         ! set icon array to 1
        subl    #1,d5
        jne     initic
        movl    #ilocbu,a4      ! input load bus array
        movl    nlb,d5
        jbsr    arryin
        movl    #lodadm,a4      ! input load admittance array
        jbsr    arrycp
        movl    ng,d5
        movl    #igenbu,a4      ! input generator bus number
        jbsr    arryin
        movl    #gpwr,a4        ! input generator power
        jbsr    arrycp
        movl    #genadm,a4      ! input generator admittance
        jbsr    arrycp
        movl    #gain,a4        ! input generator gain
```

```
                jsr     arryrl
                movl    #h,a4           | input generator H
                jsr     arryrl
                movl    #tc,a4          | input generator time const
                jsr     arryrl
                movl    #pset,a4        | input generator power set point
                jsr     arryrl
                movl    #fset,a4        | input generator frequency set point
                jsr     arryrl
                movl    #deltat,a4      | input generator algorithm step length
                jsr     arryrl
                movl    #icontr,a4
initct:         movl    #0,a4@+         | set icontr=0
                subl    #1,d5
                jne     initct
                movl    #itime,a4       | zero time
                movl    #0,a4@+
                movl    nb,d4
                movl    nb,d5
                mulu    d4,d5
                movl    #dmat,a4
                jsr     arrycp          | input d matrix
                movl    ng,d4
                movl    nb,d5
                mulu    d4,d5
                movl    #bmat,a4
                jsr     arrycp          | input b matrix
                movl    #cmat,a4
                jsr     arrycp          | input c matrix
                movl    ng,d4
                movl    ng,d5
                mulu    d4,d5
                movl    #amat,a4
                jsr     arrycp          | input a matrix
                rts

arryrl:         movl    d5,d4           | input an array of real numbers
arrilp:         jsr     nexin           | length of array in d5 in location
                movl    a2@,d0          | pointed to by a4
                jsr     be2iee
                movl    d2,a2@          | convert from b.e. to ieee format
                subl    #1,d5           | decrement counter
                jne     arrilp
                movl    d4,d5
                rts

arryin:         movl    d5,d4           | input an array of integers length d5
arinlp:         jsr     nexin           | stored at a4
                subl    #1,d5
                jne     arinlp          | decrement counter and loop
                movl    d4,d5
                rts

arrycp:         jsr     arryrl          | input complex array by 2 calls to real
                jsr     arryrl          | array routine
                rts
```

```
hexin:  jsr     gpenum
        movl    a4,a0
        movl    a4,a2
        movl    d0,a4@+
        movl    #4,d1
        trap    #15
        .word   11
        movl    #next,a2
        trap    #15
        .word   10
        rts

stcoc:  movl    #stinp,a5       | writes a command to host to start
stnex:  movb    a5@+,d0         | data transfer
        jeq     retec
        trap    #15
        .word   4
        jra     stnex
retec:  trap    #15
        .word   3
        andb    #0x7f,d0
        cmpb    #0x23,d0        | waits for # character to be sent bef
        jne     retec           | returning to caller
        rts


cutoeh: movl    a0,a1
        movl    #4,d1
nexno:  movb    a0@+,d0
        andb    #0xf0,d0
        lsrb    #4,d0
        addb    #0x30,d0
        cmpb    #0x39,d0
        jle     numer
        addb    #0x07,d0
numer:  trap    #15
        .word   2
        movb    a0@+,d0
        andb    #0x0f,d0
        addb    #0x30,d0
        cmpb    #0x39,d0
        jle     numer1
        addb    #0x07,d0
numer1: trap    #15
        .word   2
        subl    #1,d1
        jne     nexno
        movl    #0x0d,d0
        trap    #15
        .word   2
        movl    #0x0a,d0
        trap    #15
        .word   2
```

```
            rts

gpenum:     movl    #0,d7
gchar0:     trap    #15
            .word   3
            andl    #0x0000007f,d0
            subb    #0x30,d0
            jmi     gchar0
            cmpb    #0x09,d0
            jle     gotit
            subb    #0x07,d0
            cmpb    #0x0f,d0
            jle     gotit
            jra     gchar0
gchar1:     trap    #15
            .word   3
            andl    #0x0000007f,d0
            cmpb    #0x7f,d0
            jeq     gchar1
            subb    #0x30,d0
            jmi     endnum
            cmpb    #0x09,d0
            jle     gotit
            subb    #0x07,d0
            cmpb    #0x0f,d0
            jle     gotit
endnum:     movl    d7,d0
            rts
gotit:      lsll    #4,d7
            andb    #0x0f,d0
            orb     d0,d7
            jra     gchar1


gtenum:     movl    #0,d7
gthar0:     trap    #15
            .word   1
            andl    #0x0000007f,d0
            subb    #0x30,d0
            jmi     gthar0
            cmpb    #0x09,d0
            jle     gttit
            subb    #0x07,d0
            cmpb    #0x0f,d0
            jle     gttit
            jra     gthar0
gthar1:     trap    #15
            .word   1
            andl    #0x0000007f,d0
            cmpb    #0x7f,d0
            jeq     gthar1
            subb    #0x30,d0
            jmi     etdnum
            cmpb    #0x09,d0
            jle     gttit
            subb    #0x07,d0
```

```
        cmpb    #0x0f,d0
        jle     gttit
etdnum: movl    d7,d0
        rts
gttit:  lsll    #4,d7
        andb    #0x0f,d0
        orb     d0,d7
        jra     gtnar1

genint:                                 | initialise generator programs
        movl    #slgeno,a6              | a6 point to slave number
        movl    #0,d7                   | d7 holds gen no
ngenin: movl    a6@+,d6                 | d6 holds slave number
        movl    d7,d5
        asll    #2,d5                   | d5 holds gen no*4
        movl    d5,d4
        asll    #1,d4                   | d4 holds gen no*8
        movl    #gpwr,a0
        jbsr    send8                   | send gpwr for current gen no
        movl    #gain,a0
        jbsr    send4                   | send gain for current gen no
        movl    #deltat,a0
        jbsr    send4                   | send deltat for current gen no
        movl    #h,a0
        jbsr    send4                   | send h for current gen no
        movl    #tc,a0
        jbsr    send4                   | send tc for current gen no
        movl    #pset,a0
        jbsr    send4                   | send pset for current gen no
        movl    #fset,a0
        jbsr    send4                   | send fset for current gen no
        movl    #genadm,a0
        jbsr    send8                   | send genadm for current gen no
        movl    #igenbu,a0
        jbsr    send4                   | send igenbus for current gen no
        movl    #igenbu,a0
        movl    a0@(0,d5:l),d4
        subl    #1,d4
        asll    #2,d4
        movl    #busvol,a0
        jbsr    send8                   | send busvol for current igenbus
        addl    #1,d7
        cmpl    #g,d7
        jne     ngenin
        jra     gen                     | automatically do gen

netint:                                 | initialise network programs
        movl    #slbsno,a6              | a6 points to no. of busses for slave
        movl    #0,d5                   | d6 holds slave number
        movl    #0,d7                   | d7 holds mybusl
nnetin: movl    a6@,d5
        addl    d7,d5                   | d5 holds mybus2
        movl    d7,temp
        movl    #temp,a0
        movl    d6,d0
        movl    #4,d1
```

```
                jsr     ctscom          | output mybus1
                movl    d5,temp
                movl    #temp,a0
                movl    d6,d0
                movl    #4,d1
                jsr     ctscom          | output mybus2
                movl    #ng,a0
                movl    d6,d0
                movl    #4,d1
                jbsr    ctscom          | output ng
                movl    #nb,a0
                movl    d6,d0
                movl    #4,d1
                jbsr    ctscom          | output nb
                movl    #igenbu,a0
                movl    ng,d1
                asll    #2,d1
                movl    d6,d0
                jsr     ctscom          | output entire igenbu array
                movl    #genadm,a0
                movl    ng,d1
                asll    #3,d1
                movl    d6,d0
                jsr     ctscom          | output entire genadm array
                movl    #dmat,a0
                movl    d7,d3
                asll    #3,d3
                movl    nb,d1
                mulu    d1,d3
                addl    d3,a0           | get start of relevant dmat in a0
                movl    a6@+,d2
                mulu    d2,d1
                asll    #3,d1           | length = nb*(no. bus in this area)*8
                movl    d6,d0
                jsr     ctscom          | output relevant part of dmat
                movl    d5,d7
                addl    #1,d6
                cmpl    nslave,d6
                jne     nmatin
                jra     net             | automatically do network calc

gen:                                    | send icont and busval
                movl    #slgenb,a6      | a6 point to slave number
                movl    #0,d7           | d7 holds gen no
ngen:           movl    a6@+,d6         | d6 holds slave number
                movl    d7,d5
                asll    #2,d6           | d6 holds gen no*4
                movl    d5,d4
                asll    #1,d4           | d4 holds gen no*8
                movl    #icontr,a0
                jsr     send4           | send icontr for current gen no
                movl    #icontr,a0
                addl    d5,a0
                movl    #0,a0@
                movl    d4,d0
```

```
                movl    #igenbu,a0
                movl    a0@(0,d5:l),d4
                subl    #1,d4
                asll    #2,d4
                movl    #busvol,a0
                jbsr    send8           | send busvol for current igenbus
                movl    d3,d4
                movl    #icontr,a0
                addl    d5,a0
                movl    a0@,d3
                jeq     nosetb          | go for next gen if icontr=0
                movl    #bset,a0
                jbsr    send4           | send bset for current gen no
                movl    #fset,a0
                jbsr    send4           | send fset for current gen no
noseto: addl    #1,d7
                cmpl    ng,d7
                jne     ngen            | do next gen

                movl    #slgeno,a6      | a6 point to slave number
                movl    #0,d7           | d7 holds gen no
ngenge: movl    a6@+,d6         | d6 holds slave number
                movl    d7,d4
                asll    #3,d4           | d4 holds gen no*8
                movl    #gpwr,a0
                jbsr    rece8           | receive gen power
                movl    #iti,a0
                jbsr    rece8           | receive gen current
                movl    #edash,a0
                jbsr    rece8           | receive gen edash
                addl    #1,d7
                cmpl    ng,d7
                jne     ngenge          | do next generator
                rts

net:
                movl    #0,d6           | d6 holds slave number
nnet:   movl    #edash,a0
                movl    d6,d0
                movl    ng,d1
                asll    #3,d1
                jbsr    ctscom          | send edash values
                addl    #1,d6
                cmpl    nslave,d6
                jne     nnet            | do next slave

                movl    #slbsno,a6      | a6 points to no. of busses for slave
                movl    #0,d6           | d6 holds slave number
                movl    #0,d7           | d7 holds mybus1
nnet2:  movl    a6@+,d5
                addl    d7,d5           | d5 holds mybus2
                movl    #busvol,a0
                movl    d7,d3
                asll    #3,d3
                addl    d3,a0           | a0 points to busvol(mybus1)
                movl    a6@+,d1
```

```
                asll    #3,d1           | d1 holds number of bytes
                movl    d6,d0           | d0 holds slave number
                jsr     stccom          | get busvol from slave
                movl    d5,d7
                addl    #1,d6
                cmpl    nslave,d6
                jne     nnet2           | do next slave

                rts

ncont:                                  | send v1,v2,iel,ie2
tmplab:         jsr     allis           | set all slaves to listen
                movl    #iel,a0
                jsr     stallc          | output iel
tmpla2:         cmpl    #0xffffffff,iel | if ffffffff then no change: return
                jeq     rthcon
                movl    #ie2,a0
                jsr     stallc          | output ie2
                movl    #v1,a0
                jsr     stallc          | output v1
                movl    #v1+4,a0
                jsr     stallc          | output v1+4
                movl    #v2,a0
                jsr     stallc          | output v2
                movl    #v2+4,a0
                jsr     stallc          | output v2+4
                jsr     unlat           | release slaves
                jsr     stallc          | let slave containing iel talk to all
                rts

rthcon:         jsr     unlat           | release slaves
                rts


send4:          addl    d5,a0
                movl    #4,d1
                movl    d6,d0
                jra     stscom

send8:          addl    d4,a0
                movl    #8,d1
                movl    d6,d0
                jra     stscom

rece8:          addl    d4,a0
                movl    d6,d0
                jra     stccom

allis:          movl    #0,d3           | set all slave to listen
allli2:         movl    d3,d0
                jsr     setin
                addl    #1,d3
                cmpl    nslave,d3
                bne     allli2
                jsr     conot
                rts
```

```
ctallc:                                 ; controller outputs 4 bytes
        movb    a0@+,d0
        jbsr    boutout
        movb    a0@+,d0
        jbsr    boutout
        movb    a0@+,d0
        jbsr    boutout
        movb    a0@+,d0
        jbsr    bsortn
        rts

stallc:                                 ; one slave to all(including controller)
        movl    #0,d3
        movl    #0xffffffff,d7
        movl    #slbsno,a0
stall2: addl    a0@+,d3
        addl    #1,d7
        cmpl    iel,d3
        ble     stall2
        movl    #1,d6
again3: movl    #0,d3
stall3: movl    d3,d0
        cmpl    d3,d7
        jeq     incsta
        jbsr    setin
incsta: addl    #1,d3
        cmpl    nslave,d3
        jne     stall3
        movl    d7,d0
        movl    #s,a0
        jbsr    stccom
        subl    #1,d6
        jeq     again3
        rts
```

```
#include <stdio.h>
#include "fou.h"

/*     test program for network calculations */

        float busvr[128],dmatr[16384],genadr[25],edasnr[25];
        float busvi[128],dmati[16384],genadi[25],edasni[25];
        float *p2r,*p2i,*p3r,*p3i;
        int igenbus[25];
        int nb=30;
        int ng=6;
        int nl=1000;
        int h,i,k;
        char rub;

/*     main loop to calculate busvol     */

main()
{
register j,tp2r,tp3r,tir,tii;
register float *p4r,*p4i,*p1r,*p1i;

/* initialise fou */
setwtl();
for(h=0;h<ng;h++)
        {
        igenbus[h]=h;
        }
printf("start watch and press return\n");
scanf(" %c ", &rub);
for(h=0;h<nl;h++)
        {
        p1r=busvr;
        p1i=busvi;
        for(i=0;i<nb;i++)
                {
                MSD(*p1r) = 0;
                MSD(*p1i) = 0;
                p1r++;
                p1i++;
                }
                p2r=genadr;
                p2i=genadi;
                p3r=edasnr;
                p3i=edasni;
        for(i=0;i<ng;i++)
                {
                k=(i << 7);
                p4r = &dmatr[k];
                p4i = &dmati[k];
                p1r = busvr;
                p1i = busvi;
                tp2r= MSD(*p2r);
                tp3r= MSD(*p3r);
                for(j=0;j<nb;j++)
                        {
```

```
/* complex mult dmat(i,k)*genaam(j) */

l1:     FFPU( SEQ6 , LAS , 0 ) = *p4r;
        FFPU( SEQ6 ,LBSF , 0 ) = tp2r;
        FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ6 , LAS , 0 ) = *p2i;
        FFPU( SEQ6 ,LBSF , 0 ) = *p4i;

l3:
        FFPU( SEQ4 , LBSF, SS) = FFPU( SEQ7 , UNLOAD , 0 );
l2:     t1r = FFPU( SEQ5 , UNLOAD , 0 );

l4:
        FFPU( SEQ6 ,LBSF , 0 ) = *p4r;
        FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
l5:     FFPU( SEQ6 , LAS , 0 ) = *p4i;
        FFPU( SEQ6 ,LBSF , 0 ) = tp2r;
        FFPU( SEQ4 , LBSF, SA) = FFPU( SEQ7 , UNLOAD , 0 );
l6:

        t1i = FFPU( SEQ5 , UNLOAD , 0 );

/*  complex multiply t1*edasn(j)   */

l7:     FFPU( SEQ6 , LAS , 0 ) = t1r;
        FFPU( SEQ6 ,LBSF , 0 ) = tp3r;
        FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
l8:     FFPU( SEQ6 , LAS , 0 ) = *p3i;
        FFPU( SEQ6 ,LBSF , 0 ) = t1i;
        FFPU( SEQ4 , LBSF, SS) = FFPU( SEQ7 , UNLOAD , 0 );
l9:

        FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );
ld:
        FFPU( SEQ4 , LBSF, SA) = *p1r;
        *p1r = FFPU( SEQ5 , UNLOAD , 0 );

la:
        FFPU( SEQ6 ,LBSF , 0 ) = t1r;
        FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
lb:     FFPU( SEQ6 , LAS , 0 ) = t1i;
        FFPU( SEQ6 ,LBSF , 0 ) = tp3r;
        FFPU( SEQ4 , LBSF, SA) = FFPU( SEQ7 , UNLOAD , 0 );
lc:

        FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );

le:
        FFPU( SEQ4 , LBSF, SA) = *p1i;
        *p1i = FFPU( SEQ5 , UNLOAD , 0 );
```

```
        /* increment bus pointers */

                                p1r++;
                                p1i++;
                                p4r++;
                                p4i++;
                                }

        /*  increment generator pointers  */

                        g2r++;
                        g2i++;
                        g3r++;
                        g3i++;
                        }
                }
        printf(" stop timing \n" );
        if( j < 0 )
                {
                if( j == -1 ) goto l1;
                if( j == -2 ) goto l2;
                if( j == -3 ) goto l3;
                if( j == -4 ) goto l4;
                if( j == -5 ) goto l5;
                if( j == -6 ) goto l6;
                if( j == -7 ) goto l7;
                if( j == -8 ) goto l8;
                if( j == -9 ) goto l9;
                if( j == -10) goto la;
                if( j == -11) goto lb;
                if( j == -12) goto lc;
                if( j == -13) goto ld;
                if( j == -14) goto le;
                }
        }
        setwtl()
        {
        /*  set timers in both chips  */
                IFPJ( 0504 , MB0A, LMODE ) = 10 ;
                IFPJ( 0504 , MB1A, LMODE ) = 10 ;
                IFPJ( 0504 , MB2A, LMODE ) = 10 ;
                IFPJ( 0504 , MB3A, LMODE ) = 10 ;
                IFPJ( 0506 , MB0M, LMODE ) = 10 ;
                IFPJ( 0506 , MB1M, LMODE ) = 10 ;
                IFPJ( 0506 , MB2M, LMODE ) = 10 ;
                IFPJ( 0506 , MB3M, LMODE ) = 10 ;
                cache();
        }
        ioctl()
        {
        }
```

```
nclude <stdio.h>
#include "fou.h"

/* test program for generator calculations */

        float genadr,iitr,ietr,iedasnr,oedasnr[25],o;enadr[25];
        float genadi,iiti,ieti,iedasni,oedashi[25],o;genadi[25];
        float dmatr[16384],dmati[16384];
        float be,bn,ih,itc,igain,ideltat,ideltat;
        float wit,bi2,pifrq,pifq2,psto,fstp,didt;
        float dito,dwdt,wito,dpmdt,pmtp;
        float sintab[4096],costab[4096];
        float index = 4096;
        float tempr,tempi;
        float edasnmag,PI;
        float idltt2;
        float *p4r,*p4i;
        int *p1;
        int igenbus[25],mybus;
        int ng=6;
        int nl=1000;
        char rub;


main()
{
register i,n,k;
register float *p2r,*p2i,*p3r,*p3i;

/* assume initialisation has been done , timing only for running loop */
/* use genadr not genimp, invert ih bi2,itc beforhand to avoid divide */
/* idltt2 = ideltat/2, all this is set up by initialise */
for(i=0;i<ng;i++)
        {
        igenbus[i]=i;
        }
setwtl();
printf("start watch and press return\n");
scanf(" %c ", &rub );

for(n=0;n<nl;n++)
        {

/* iiti = (iedasn-iet)*genadn */

        FEPU( SEQ4 , LAS , 0 ) = iedasni;
        FEPU( SEQ4 , LBSF, SS) = ieti;
I61:    tempi = FEPU( SEQ5 , UNLOAD , 0 );
        FEPU( SEQ4 , LAS , 0 ) = iedasnr;
        FEPU( SEQ4 , LBSF, SS) = ietr;
I1:     FEPU( SEQ6 , LAS , 0 ) = FEPU( SEQ5 , UNLOAD , 0 );

        FEPU( SEQ6 ,LBSF , 0 ) = genadr;
I2:     FEPU( SEQ4 , LAS , 0 ) = FEPU( SEQ7 , UNLOAD , 0 );
        FEPU( SEQ6 , LAS , 0 ) = tempi;
        FEPU( SEQ6 ,LBSF , 0 ) = genadi;
```

```
13:     FFPU( SEQ4 , LBSF, SS) = FFPU( SEQ7 , UNLOAD , 0 );
162:    iitr = FFPU( SEQ5 , UNLOAD , 0 );
        FFPU( SEQ6 ,LBSF , 0 ) = genadr;
14:     FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ6 , LAS , 0 ) = tempr;
        FFPU( SEQ6 ,LBSF , 0 ) = genadi;
15:     FFPU( SEQ4 , LBSF, SA) = FFPU( SEQ7 , UNLOAD , 0 );
163:    iiti = FFPU( SEQ5 , UNLOAD , 0 );

/* de = REAL(iiti*CONJG(edasn)) */


        FFPU( SEQ6 , LAS , 0 ) = iitr;
        FFPU( SEQ6 ,LBSF , 0 ) = iedasnr;
16:     FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ6 , LAS , 0 ) = iiti;
        FFPU( SEQ6 ,LBSF , 0 ) = iedasni;
17:     FFPU( SEQ4 , LBSF, SA) = FFPU( SEQ7 , UNLOAD , 0 );
164:    de = FFPU( SEQ5 , UNLOAD , 0 );

/* dimt=wit-difq2 */

        FFPU( SEQ4 , LAS , 0 ) = wit;
        FFPU( SEQ4 , LBSF, SS) = difq2;
165:    dimt = FFPU( SEQ5 , UNLOAD , 0 );

/* dwdt = difq*((pm-pe)/ih)) ih has been inverted so *ih!!!! */

        FFPU( SEQ4 , LAS , 0 ) = pm;
        FFPU( SEQ4 , LBSF, SS) = pe;
18:     FFPU( SEQ6 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );
        FFPU( SEQ6 , LBSF, 0 ) = ih;
19:     FFPU( SEQ6 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ6 , LBSF, 0 ) = difq;
166:    dwdt = FFPU( SEQ7 , UNLOAD , 0 );

/* dpmdt = (pstp+igain*(fstp-(wit/pi2))-pm)*itc    itc and pi2 already inv */

        FFPU( SEQ4 , LAS , 0 ) = fstp;
        FFPU( SEQ6 , LAS , 0 ) = wit;
        FFPU( SEQ6 , LBSF, 0 ) = pi2;
110:    FFPU( SEQ4 , LBSF, SS) = FFPU( SEQ7 , UNLOAD , 0 );
111:    FFPU( SEQ6 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );
        FFPU( SEQ6 , LAS , 0 ) = igain;
112:    FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ4 , LBSF, SA) = pstp;
113:    FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );
        FFPU( SEQ4 , LBSF, SS) = pm;
114:    FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );
        FFPU( SEQ6 , LBSF, 0 ) = itc;
167:    dpmdt = ( SEQ7 , UNLOAD , 0 );

/* pmtp = pm+(dpmdt*deltat) */

        FFPU( SEQ5 , LAS , 0 ) = ideltat;
        FFPU( SEQ6 , LBSF, 0 ) = dpmdt;
```

```
194:    FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ4 , LBSF, SA) = pm;
168:    pmtp = FFPU( SEQ5 , UNLOAD , 0 );

/* witp = wit + (dwdt*ideltat) */

        FFPU( SEQ6 , LBSF, 0 ) = dwdt;
115:    FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ4 , LBSF, SA) = wit;
169:    witp = FFPU( SEQ5 , UNLOAD , 0);

/* ditp = idelta+(didt*ideltat) */

        FFPU( SEQ6 , LBSF, 0 ) = didt;
116:    FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ4 , LBSF, SA) = idelta;
170:    ditp = FFPU( SEQ5 , UNLOAD , 0 );

/* convert an angle to look up table index */

        FFPU( SEQ6 , LAS , 0 ) = ditp;
        FFPU( SEQ6 , LBSF, 0 ) = pi2;
171:    tempr = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ4 , LASF,C2IS ) = tempr;
        FFPU( SEQ4 , LASF, SF) = FFPU( SEQ5 , 0 , 0 );
172:    tempi = FFPU( SEQ5 , UNLOAD , 0 );
        FFPU( SEQ4 , LAS , 0 ) = tempr;
        FFPU( SEQ4 , LBSF, SS) = tempi;
117:    FFPU( SEQ6 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );
        FFPU( SEQ6 , LBSF, 0 ) = index;
118:    FFPU( SEQ4 , LASF,C2IS)= FFPU( SEQ7 , UNLOAD , 0 );
        i = FFPU( SEQ5 , 0 , 0 );

/* i can be used to index sintab and costab */

/* iedasn = cmplx(edashmag*cos(ditp),edashmag*sin(ditp)); */

        FFPU( SEQ6 , LAS , 0 ) = edashmag;
        FFPU( SEQ6 , LBSF, 0 ) = costab[i];
173:    iedasnr = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ6 , LBSF, 0 ) = sintab[i];
174:    iedashi = FFPU( SEQ7 , UNLOAD , 0 );

/* now recalculate the bus voltage iet */
        b1=igenbus;
        b2r=edashr;
        b2i=edashi;
        b3r=edashdr;
        b3i=edashdi;
        *(SD( ieti ) = 0;
        *(SD( ietr ) = 0;
        for(i=0;i<ng;i++)
                {
                k = *b1;
                k=( k << 7 ) & mybus;
                b4r = admatr[k];
```

```
                    p4i = &dmati[k];

            FFPU( SEQ6 , LAS , 0 ) = *p3r;
            FFPU( SEQ6 ,LBSF , 0 ) = *p2r;
119:        FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
            FFPU( SEQ6 , LAS , 0 ) = *p3i;
            FFPU( SEQ6 ,LBSF , 0 ) = *p2i;
120:        FFPU( SEQ4 , LBSF, SS) = FFPU( SEQ7 , UNLOAD , 0 );
175:        tempr = FFPU( SEQ5 , UNLOAD , 0 );

            FFPU( SEQ6 ,LBSF , 0 ) = *p2r;
121:        FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
            FFPU( SEQ6 , LAS , 0 ) = *p3r;
            FFPU( SEQ6 ,LBSF , 0 ) = *p2i;
122:        FFPU( SEQ4 , LBSF, SA) = FFPU( SEQ7 , UNLOAD , 0 );
176:        tempi = FFPU( SEQ5 , UNLOAD , 0 );


            FFPU( SEQ6 , LAS , 0 ) = tempr;
            FFPU( SEQ6 ,LBSF , 0 ) = *p4r;
123:        FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
            FFPU( SEQ6 , LAS , 0 ) = tempi;
            FFPU( SEQ6 ,LBSF , 0 ) = *p4i;
124:        FFPU( SEQ4 , LBSF, SS) = FFPU( SEQ7 , UNLOAD , 0 );
125:        FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );
            FFPU( SEQ4 , LBSF, SA) = ietr;
177:        ietr = FFPU( SEQ5 , UNLOAD , 0 );

            FFPU( SEQ6 ,LBSF , 0 ) = *p4r;
126:        FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
            FFPU( SEQ6 , LAS , 0 ) = tempr;
            FFPU( SEQ6 ,LBSF , 0 ) = *p4i;
127:        FFPU( SEQ4 , LBSF, SA) = FFPU( SEQ7 , UNLOAD , 0 );
128:        FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );
            FFPU( SEQ4 , LBSF, SA) = ieti;
179:        ieti = FFPU( SEQ5 , UNLOAD , 0 );

            p1++;
            p2i++;
            p2r++;
            p3i++;
            p3r++;
            }
        /* iiti = (iedasn-iet)*genadn */

            FFPU( SEQ4 , LAS , 0 ) = iedasni;
            FFPU( SEQ4 , LBSF, SS) = ieti;
190:        tempi = FFPU( SEQ5 , UNLOAD , 0 );
            FFPU( SEQ4 , LAS , 0 ) = iedasnr;
            FFPU( SEQ4 , LBSF, SS) = ietr;
199:        FFPU( SEQ6 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );

            FFPU( SEQ4 ,LBSF , 0 ) = genadr;
190:        FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
            FFPU( SEQ6 , LAS , 0 ) = tempi;
            FFPU( SEQ6 ,LBSF , 0 ) = genadi;
```

```
131:    FFPU( SEQ4 , LBSF, SS) = FFPU( SEQ7 , UNLOAD , 0 );
181:    iitr = FFPU( SEQ5 , UNLOAD , 0 );
        FFPU( SEQ6 ,LBSF , 0 ) = genadr;
132:    FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ6 , LAS , 0 ) = tempr;
        FFPU( SEQ6 ,LBSF , 0 ) = genadi;
133:    FFPU( SEQ4 , LBSF, SA) = FFPU( SEQ7 , UNLOAD , 0 );
182:    iiti = FFPU( SEQ5 , UNLOAD , 0 );

/* oe = REAL(iiti*CONJG(edasn)) */


        FFPU( SEQ6 , LAS , 0 ) = iitr;
        FFPU( SEQ6 ,LBSF , 0 ) = iedasnr;
134:    FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ6 , LAS , 0 ) = iiti;
        FFPU( SEQ6 ,LBSF , 0 ) = iedashi;
135:    FFPU( SEQ4 , LBSF, SA) = FFPU( SEQ7 , UNLOAD , 0 );
183:    oe = FFPU( SEQ5 , UNLOAD , 0 );

/* idelta = idelta+(diut+witp-pifq2)*idltt2 */

        FFPU( SEQ4 , LAS , 0 ) = witp;
        FFPU( SEQ4 , LBSF, SS) = pifq2;
136:    FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 ),
        FFPU( SEQ4 , LBSF, SA) = diut;
137:    FFPU( SEQ6 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );
        FFPU( SEQ6 , LBSF , 0) = idltt2;
138:    FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ6 , LBSF, 0 ) = idelta;
184:    idelta =FFPU( SEQ7 , UNLOAD , 0 );

/* wit = wit+(dwit+(((pmto-oe)*pifrq)/ih))*idltt2  remember ih is inverted */

        FFPU( SEQ4 , LAS , 0 ) = pmtp;
        FFPU( SEQ4 , LBSF, SS) = oe;
139:    FFPU( SEQ6 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );
        FFPU( SEQ6 , LBSF, 0 ) = pifrq;
140:    FFPU( SEQ6 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ6 , LBSF, 0 ) = in;
141:    FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ4 , LBSF, SA) = dwit;
142:    FFPU( SEQ6 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );
        FFPU( SEQ6 , LBSF, 0 ) = idltt2;
143:    FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ4 , LBSF, SA) = wit;
185:    wit = FFPU( SEQ7 , UNLOAD , 0 );

/* pmtpm+((dpmt+(psto+(igain*(fstp-(witp/pi2))))-pmto)/itc)*idltt2) */

        FFPU( SEQ4 , LAS , 0 ) = fstp;
        FFPU( SEQ6 , LAS , 0 ) = witp;
        FFPU( SEQ6 , LBSF, 0 ) = pi2;
144:    FFPU( SEQ4 , LBSF, SS) = FFPU( SEQ7 , UNLOAD , 0 );
145:    FFPU( SEQ6 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );
        FFPU( SEQ6 , LBSF, 0 ) = igain;
```

```
146:    FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ4 , LBSF, SS) = omtp;
147:    FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );
        FFPU( SEQ4 , LBSF, SA) = ostp;
148:    FFPU( SEQ6 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );
        FFPU( SEQ6 , LBSF, 0 ) = itc;
149:    FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ4 , LBSF, SA) = dpmdt;
150:    FFPU( SEQ6 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );
        FFPU( SEQ6 , LBSF, 0 ) = idltt2;
151:    FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ4 , LBSF, SA) = pm;
186:    pm = FFPU( SEQ5 , UNLOAD , 0 );

/* iedasn = CMPLX( iedasnmag*cos(idelta), iedasnmag*cos( idelta) ) */
/* convert an angle to look up table index */

        FFPU( SEQ6 , LAS , 0 ) = idelta;
        FFPU( SEQ6 , LBSF, 0 ) = pi2;
187:    tempr = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ4 , LASF,C21S ) = tempr;
        FFPU( SEQ4 , LASF, SF) = FFPU( SEQ5 , 0 , 0 );
188:    tempi = FFPU( SEQ5 , UNLOAD , 0 );
        FFPU( SEQ4 , LAS , 0 ) = tempr;
        FFPU( SEQ4 , LBSF, SS) = tempi;
152:    FFPU( SEQ6 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );
        FFPU( SEQ6 , LBSF, 0 ) = index;
153:    FFPU( SEQ4 , LASF,C21S)= FFPU( SEQ7 , UNLOAD , 0 );
        i = FFPU( SEQ5 , 0 , 0 );

/* i can be used to index sintab and costab */


        FFPU( SEQ6 , LAS , 0 ) = edasnmag;
        FFPU( SEQ6 , LBSF, 0 ) = costab[i];
189:    iedasnr = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ6 , LBSF, 0 ) = sintab[i];
190:    iedasni = FFPU( SEQ7 , UNLOAD , 0 );

/* iiti = (iedasn-iet)*penadm */

        FFPU( SEQ4 , LAS , 0 ) = iedasni;
        FFPU( SEQ4 , LBSF, SS) = ieti;
191:    tempi = FFPU( SEQ5 , UNLOAD , 0 );
        FFPU( SEQ4 , LAS , 0 ) = iedasnr;
        FFPU( SEQ4 , LBSF, SS) = ietr;
154:    FFPU( SEQ6 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );

        FFPU( SEQ5 ,LBSF , 0 ) = penadr;
155:    FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ6 , LAS , 0 ) = tempi;
        FFPU( SEQ6 ,LBSF , 0 ) = penadi;
156:    FFPU( SEQ4 , LBSF, SS) = FFPU( SEQ7 , UNLOAD , 0 );
178:    iitr = FFPU( SEQ5 , UNLOAD , 0 );
        FFPU( SEQ6 ,LBSF , 0 ) = penadr;
157:    FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
```

```
        FFPU( SEQ6 , LAS , 0 ) = tempr;
        FFPU( SEQ6 ,LBSF , 0 ) = genadi;
158:    FFPU( SEQ4 , LBSF, SA) = FFPU( SEQ7 , UNLOAD , 0 );
192:    iiti = FFPU( SEQ5 , UNLOAD , 0 );

/* be = REAL(iiti*CONJG(iedasn)) */


        FFPU( SEQ6 , LAS , 0 ) = iitr;
        FFPU( SEQ6 ,LBSF , 0 ) = iedasnr;
159:    FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ6 , LAS , 0 ) = iiti;
        FFPU( SEQ6 ,LBSF , 0 ) = iedashi;
160:    FFPU( SEQ4 , LBSF, SA) = FFPU( SEQ7 , UNLOAD , 0 );
193:    be = FFPU( SEQ5 , UNLOAD , 0 );

        }
printf(" stop timing \n");
if( n < 0)
        {

        if( n == -1) goto 11;
        if( n == -2) goto 12;
        if( n == -3) goto 13;
        if( n == -4) goto 14;
        if( n == -5) goto 15;
        if( n == -6) goto 16;
        if( n == -7) goto 17;
        if( n == -8) goto 18;
        if( n == -9) goto 19;
        if( n == -10) goto 110;
        if( n == -11) goto 111;
        if( n == -12) goto 112;
        if( n == -13) goto 113;
        if( n == -14) goto 114;
        if( n == -15) goto 115;
        if( n == -16) goto 116;
        if( n == -17) goto 117;
        if( n == -18) goto 118;
        if( n == -19) goto 119;
        if( n == -20) goto 120;
        if( n == -21) goto 121;
        if( n == -22) goto 122;
        if( n == -23) goto 123;
        if( n == -24) goto 124;
        if( n == -25) goto 125;
        if( n == -26) goto 126;
        if( n == -27) goto 127;
        if( n == -28) goto 128;
        if( n == -29) goto 129;
        if( n == -30) goto 130;
        if( n == -31) goto 131;
        if( n == -32) goto 132;
        if( n == -33) goto 133;
        if( n == -34) goto 134;
        if( n == -35) goto 135;
```

```
if( n == -35) goto 135;
if( n == -37) goto 137;
if( n == -38) goto 138;
if( n == -39) goto 139;
if( n == -40) goto 140;
if( n == -41) goto 141;
if( n == -42) goto 142;
if( n == -43) goto 143;
if( n == -44) goto 144;
if( n == -45) goto 145;
if( n == -46) goto 146;
if( n == -47) goto 147;
if( n == -48) goto 148;
if( n == -49) goto 149;
if( n == -50) goto 150;
if( n == -51) goto 151;
if( n == -52) goto 152;
if( n == -53) goto 153;
if( n == -54) goto 154;
if( n == -55) goto 155;
if( n == -56) goto 156;
if( n == -57) goto 157;
if( n == -58) goto 158;
if( n == -59) goto 159;
if( n == -60) goto 160;
if( n == -61) goto 161;
if( n == -62) goto 162;
if( n == -63) goto 163;
if( n == -64) goto 164;
if( n == -65) goto 165;
if( n == -66) goto 166;
if( n == -67) goto 167;
if( n == -68) goto 168;
if( n == -69) goto 169;
if( n == -70) goto 170;
if( n == -71) goto 171;
if( n == -72) goto 172;
if( n == -73) goto 173;
if( n == -74) goto 174;
if( n == -75) goto 175;
if( n == -76) goto 176;
if( n == -77) goto 177;
if( n == -78) goto 178;
if( n == -79) goto 179;
if( n == -80) goto 180;
if( n == -81) goto 181;
if( n == -82) goto 182;
if( n == -83) goto 183;
if( n == -84) goto 184;
if( n == -85) goto 185;
if( n == -86) goto 186;
if( n == -87) goto 187;
if( n == -88) goto 188;
if( n == -89) goto 189;
if( n == -90) goto 190;
if( n == -91) goto 191;
```

```
            if( n == -92) goto l92;
            if( n == -93) goto l93;
            if( n == -94) goto l94;
            }
}
setwtl()
{
/*  set timers in both chips  */
            IFPU( SEQ4 , MB0A, LMODE ) = 10 ;
            IFPU( SEQ4 , MB1A, LMODE ) = 10 ;
            IFPU( SEQ4 , MB2A, LMODE ) = 10 ;
            IFPU( SEQ4 , MB3A, LMODE ) = 10 ;
            IFPU( SEQ6 , MB0M, LMODE ) = 10 ;
            IFPU( SEQ6 , MB1M, LMODE ) = 10 ;
            IFPU( SEQ6 , MB2M, LMODE ) = 10 ;
            IFPU( SEQ6 , MB3M, LMODE ) = 10 ;
            cache();
}
ioctl()
{
}
```

```
#include <stdio.h>
#include "fpu.h"

/*     test program for householder calculations */

        float *p1i,*p2i;
        float dmatr[16384],v1r,v2r,cr[128],dr[128];
        float dmati[16384],v1i,v2i,ci[128],di[128];
        float unitr=1.0;
        int sono=0;
        int np=30;
        int nl=1000;
        int npa=30;
        int n,j,iel,ie2;
        char rub;


main()
{
register i,tempr,tempi,sr,si;
register float *p1r,*p2r,*p3r,*p3i;
iel = 5;
ie2 = 17;
setwtl();
printf(" start watch and press return \n");
scanf("%c",&rub);
/* work out s first */

/* v2*dmat(ie2,iel) */

for(n=0;n<nl;n++)
{

i = ( iel << 7 ) ;
j = i & ie2 ;
i = i & iel ;


p1r = &dmatr[j];
p1i = &dmati[j];

FPPU( SEQ6 , LAS , 0 ) = v2r;
FPPU( SEQ6 ,LBSF , 0 ) = *p1r;
11:FPPU( SEQ4 , LAS , 0 ) = FPPU( SEQ7 , UNLOAD , 0 );
FPPU( SEQ6 , LAS , 0 ) = v2i;
FPPU( SEQ6 ,LBSF , 0 ) = *p1i;
12:FPPU( SEQ4 , LBSF, 99) = FPPU( SEQ7 , UNLOAD , 0 );
13:tempr = IFPU( SEQ5 , UNLOAD , 0 );

FPPU( SEQ6 ,LBSF , 0 ) = *p1r;
14:FPPU( SEQ4 , LAS , 0 ) = FPPU( SEQ7 , UNLOAD , 0 );
FPPU( SEQ6 , LAS , 0 ) = v2r;
FPPU( SEQ6 ,LBSF , 0 ) = *p1i;
15:FPPU( SEQ4 , LBSF, 88) = FPPU( SEQ7 , UNLOAD , 0 );
16:tempi = IFPU( SEQ5 , UNLOAD , 0 );
```

```
        plr = &dmatr[i];
        pli = &dmati[i];

        FFPU( SEQ6 , LAS , 0 ) = vlr;
        FFPU( SEQ6 ,LBSF , 0 ) = *plr;
        17:FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ6 , LAS , 0 ) = vli;
        FFPU( SEQ6 ,LBSF , 0 ) = *pli;
        18:FFPU( SEQ4 , LBSF, SS) = FFPU( SEQ7 , UNLOAD , 0 );
        19:FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );
        IFPU( SEQ4 , LBSF, SA) = tempr;
        110:FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );
        FFPU( SEQ4 , LBSF, SA) = unitr;
        111:sr = IFPU( SEQ5 , UNLOAD , 0 );

        FFPU( SEQ6 ,LBSF , 0 ) = *plr;
        112:FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
        FFPU( SEQ6 , LAS , 0 ) = vlr;
        FFPU( SEQ6 ,LBSF , 0 ) = *pli;
        113:FFPU( SEQ4 , LBSF, SA) = FFPU( SEQ7 , UNLOAD , 0 );
        114:FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ5 , UNLOAD , 0 );
        IFPU( SEQ4 , LBSF, SA) = tempi;
        115:si = IFPU( SEQ5 , UNLOAD , 0 );

        /* negate s to give addition in inner loop */

        si = si ^ 0x80000000 :
        sr = sr ^ 0x80000000 :

        /* loop to produce c(i) */
        i = ( iel << 7 ) | sono;
        j = ( iez << 7 ) | sono;

        plr = &dmatr[i];
        pli = &dmati[i];
        p2r = &dmatr[j];
        p2i = &dmati[j];
        p3r = cr;
        p3i = ci;

        for(i=0;i<noa;i++)
                {

                FFPU( SEQ6 , LAS , 0 ) = *plr;
                FFPU( SEQ6 ,LBSF , 0 ) = vlr;
        116:    FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
                FFPU( SEQ6 , LAS , 0 ) = *pli;
                FFPU( SEQ6 ,LBSF , 0 ) = vli;
        117:    FFPU( SEQ4 , LBSF, SS) = FFPU( SEQ7 , UNLOAD , 0 );
        118:    tempr = IFPU( SEQ5 , UNLOAD , 0 );

                FFPU( SEQ6 ,LBSF , 0 ) = vlr;
        119:    FFPU( SEQ4 , LAS , 0 ) = FFPU( SEQ7 , UNLOAD , 0 );
                FFPU( SEQ6 , LAS , 0 ) = *plr;
                FFPU( SEQ6 ,LBSF , 0 ) = vli;
```

```
120:    FPPU( SEQ4 , LBSF, SA) = FPPU( SEQ7 , UNLOAD , 0 );
121:    tempi = IPPU( SEQ5 , UNLOAD , 0 );

        FPPU( SEQ6 , LAS , 0 ) = *p2r;
        FPPU( SEQ6 ,LBSF , 0 ) = v2r;
122:    FPPU( SEQ4 , LAS , 0 ) = FPPU( SEQ7 , UNLOAD , 0 );
        FPPU( SEQ6 , LAS , 0 ) = *p2i;
        FPPU( SEQ6 ,LBSF , 0 ) = v2i;
123:    FPPU( SEQ4 , LBSF, SS) = FPPU( SEQ7 , UNLOAD , 0 );
124:    FPPU( SEQ4 , LAS , 0 ) = FPPU( SEQ5 , UNLOAD , 0 );
        IPPU( SEQ4 , LBSF, SA) = tempr;
125:    tempr = IPPU( SEQ5 , UNLOAD , 0 );

        FPPU( SEQ6 ,LBSF , 0 ) = v2r;
126:    FPPU( SEQ4 , LAS , 0 ) = FPPU( SEQ7 , UNLOAD , 0 );
        FPPU( SEQ6 , LAS , 0 ) = *p2r;
        FPPU( SEQ6 ,LBSF , 0 ) = v2i;
127:    FPPU( SEQ4 , LBSF, SA) = FPPU( SEQ7 , UNLOAD , 0 );
128:    FPPU( SEQ4 , LAS , 0 ) = FPPU( SEQ5 , UNLOAD , 0 );
        IPPU( SEQ4 , LBSF, SA) = tempi;
129:    tempi = IPPU( SEQ5 , UNLOAD , 0 );


/* divide by s */


        IPPU( SEQ6 , LAS , 0 ) = tempr;
        IPPU( SEQ6 ,LBSF , 0 ) = sr;
130:    FPPU( SEQ4 , LAS , 0 ) = FPPU( SEQ7 , UNLOAD , 0 );
        IPPU( SEQ6 , LAS , 0 ) = tempi;
        IPPU( SEQ6 ,LBSF , 0 ) = si;
131:    FPPU( SEQ4 , LBSF, SA) = FPPU( SEQ7 , UNLOAD , 0 );
132:    *p3r = FPPU( SEQ5 , UNLOAD , 0 );

        IPPU( SEQ6 ,LBSF , 0 ) = sr;
133:    FPPU( SEQ4 , LAS , 0 ) = FPPU( SEQ7 , UNLOAD , 0 );
        IPPU( SEQ6 , LAS , 0 ) = tempr;
        IPPU( SEQ6 ,LBSF , 0 ) = si;
134:    FPPU( SEQ4 , LBSF, SS) = FPPU( SEQ7 , UNLOAD , 0 );
135:    *p3i = FPPU( SEQ5 , UNLOAD , 0 );

        IPPU( SEQ6 , LAS , 0 ) = sr;
        IPPU( SEQ6 , LBSF, 0 ) = sr;
136:    FPPU( SEQ4 , LAS , 0 ) = FPPU( SEQ7 , UNLOAD , 0 );
        IPPU( SEQ6 , LAS , 0 ) = si;
        IPPU( SEQ6 , LBSF, 0 ) = si;
137:    FPPU( SEQ4 , LBSF, SA) = FPPU( SEQ7 , UNLOAD , 0 );


138:    tempr = IPPU( SEQ5 , UNLOAD , 0 );
        FPPU( SEQ4 , LAS , 0 ) = unitr;
        IPPU( SEQ4 , LBSF, SD) = tempr;
139:    tempr = IPPU( SEQ5 , UNLOAD , 0 );

        FPPU( SEQ6 , LAS , 0 ) = *p3r;
```

```
                FPPU( SEQ6 , LBSF, 0 ) = tempr;
140:            *p3r = FPPU( SEQ7 , UNLOAD , 0 );
                FPPU( SEQ6 , LAS , 0 ) = *p3i;
                FPPU( SEQ6 , LBSF, 0 ) = tempr;
141:            *p3i = FPPU( SEQ7 , UNLOAD , 0 );

        /* increment pointers */

                p1r++;
                p2r++;
                p3r++;
                p1i++;
                p2i++;
                p3i++;
                }

        /* outer loop of matrix alteration */


        p2r = cr;
        p2i = ci;
        for(j=0;j<nba;j++)
                {
                i = ( j << 7 );
                p1r = &dmatr[i];
                p1i = &dmati[i];
                p3r = dr;
                p3i = di;

        /* inner loop */

                for(i=0;i<nb;i++)
                        {

                        FPPU( SEQ6 , LAS , 0 ) = *p2r;
                        FPPU( SEQ6 ,LBSF , 0 ) = *p3r;
142:                    FPPU( SEQ4 , LAS , 0 ) = FPPU( SEQ7 , UNLOAD , 0 );
                        FPPU( SEQ6 , LAS , 0 ) = *p2i;
                        FPPU( SEQ6 ,LBSF , 0 ) = *p2i;
143:                    FPPU( SEQ4 , LBSF, SS) = FPPU( SEQ7 , UNLOAD , 0 );

144:                    FPPU( SEQ4 , LAS , 0 ) = FPPU( SEQ5 , UNLOAD , 0 );
                        FPPU( SEQ4 , LBSF, SA) = *p1r;
145:                    *p1r = FPPU( SEQ5 , UNLOAD , 0 );

                        FPPU( SEQ6 ,LBSF , 0 ) = *p2r;
146:                    FPPU( SEQ4 , LAS , 0 ) = FPPU( SEQ7 , UNLOAD , 0 );
                        FPPU( SEQ6 , LAS , 0 ) = *p2r;
                        FPPU( SEQ6 ,LBSF , 0 ) = *p3i;
147:                    FPPU( SEQ4 , LBSF, SA) = FPPU( SEQ7 , UNLOAD , 0 );
148:                    FPPU( SEQ4 , LAS , 0 ) = FPPU( SEQ5 , UNLOAD , 0 );
                        FPPU( SEQ4 , LBSF, SA) = *p1i;
149:                    *p1i = FPPU( SEQ5 , UNLOAD , 0 );

        /* increment pointers */
```

```
                        b1r++;
                        b1i++;
                        b3r++;
                        b3i++;
                        }
                b2i++;
                b2r++;
                }
        }
        printf(" stop timing \n" );
        if( j < 0 )
                {
                if( j == -1 ) goto l1;
                if( j == -2 ) goto l2;
                if( j == -3 ) goto l3;
                if( j == -4 ) goto l4;
                if( j == -5 ) goto l5;
                if( j == -6 ) goto l6;
                if( j == -7 ) goto l7;
                if( j == -8 ) goto l8;
                if( j == -9 ) goto l9;
                if( j == -10) goto l11;
                if( j == -11) goto l12;
                if( j == -12) goto l13;
                if( j == -13) goto l14;
                if( j == -14) goto l15;
                if( j == -15) goto l16;
                if( j == -16) goto l17;
                if( j == -17) goto l18;
                if( j == -18) goto l19;
                if( j == -19) goto l20;
                if( j == -20) goto l21;
                if( j == -21) goto l22;
                if( j == -22) goto l23;
                if( j == -23) goto l24;
                if( j == -24) goto l25;
                if( j == -25) goto l26;
                if( j == -26) goto l27;
                if( j == -27) goto l28;
                if( j == -28) goto l29;
                if( j == -29) goto l30;
                if( j == -30) goto l31;
                if( j == -31) goto l32;
                if( j == -32) goto l33;
                if( j == -33) goto l34;
                if( j == -34) goto l35;
                if( j == -35) goto l36;
                if( j == -36) goto l37;
                if( j == -37) goto l38;
                if( j == -38) goto l39;
                if( j == -39) goto l40;
                if( j == -40) goto l41;
                if( j == -41) goto l42;
                if( j == -42) goto l43;
                if( j == -43) goto l44;
                if( j == -44) goto l45;
```

```
            if( j == -45) goto l45;
            if( j == -46) goto l47;
            if( j == -47) goto l48;
            if( j == -48) goto l49;
            if( j == -49) goto l10;
            }
}
setwtl()
{
/*  set timers in both cpios  */
            IFPUT( SEQ4 , MB0A, LMODE ) = 10 ;
            IFPUT( SEQ4 , MB1A, LMODE ) = 10 ;
            IFPUT( SEQ4 , MB2A, LMODE ) = 10 ;
            IFPUT( SEQ4 , MB3A, LMODE ) = 10 ;
            IFPUT( SEQ5 , MB0M, LMODE ) = 10 ;
            IFPUT( SEQ5 , MB1M, LMODE ) = 10 ;
            IFPUT( SEQ5 , MB2M, LMODE ) = 10 ;
            IFPUT( SEQ5 , MB3M, LMODE ) = 10 ;
            cache();
}
ioctl()
{
}
```