

**EVALUATION OF LOGICAL
EXTERNAL MEMORY ARCHITECTURES
FOR MULTIPROCESSOR SYSTEMS***

S. Sivaramakrishnan

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-88-32

September 1988

* This work was partially supported by the IBM Corporation under Contract Number 616513.

EVALUATION OF LOGICAL EXTERNAL
MEMORY ARCHITECTURES FOR
MULTIPROCESSOR SYSTEMS

by

S.SIVARAMAKRISHNAN, B.E.

THESIS

Presented to the Faculty of the Graduate School of
The University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

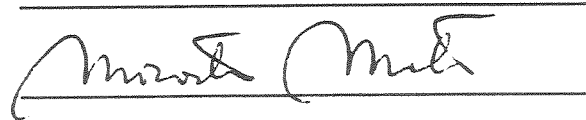
MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

August, 1988

EVALUATION OF LOGICAL EXTERNAL
MEMORY ARCHITECTURES FOR
MULTIPROCESSOR SYSTEMS

APPROVED:



A handwritten signature, possibly reading "Muzah Mub", is written between two horizontal lines.

To my wonderful parents,my sister Raji and her family

Abstract

Parallel structuring of External Memory operations has been a relatively unexplored area in Parallel Processing. This thesis effort describes an effective parallel formulation of two I/O-intensive seismic data processing algorithms on a large shared-memory multiprocessor system. Simulation modeling has been used as the vehicle for performance evaluation. Results suggest that applications which are I/O bound in most uniprocessors would probably become more compute-bound in memory-rich environments that are characteristic of large multiprocessor systems.

Table of Contents

Abstract	iv
Table of Contents	v
List of Figures	vii
1. Introduction	1
1.1 Overview	1
1.2 Problem Statement	2
1.3 Approach	3
1.4 Results	4
1.5 Organization	4
2. Seismic Data Processing	5
2.1 Overview of Migration Processing	5
2.2 Split-Step Fourier Migration	5
2.3 Phase-Shift Migration	9
3. RESQ Models for 3D Migration Processing	11
3.1 Outline of RESQ Model	11
3.2 Assumptions	13
3.3 Computation Structure	14
3.3.1 Split-Step Migration Algorithm	14
3.3.2 Phase-Shift Migration Algorithm	19

3.4 I/O System Model	22
4. Experiments and Results	28
4.1 Simulation Experiments	28
4.2 Results from Simulation Models	29
4.3 Simulation Runs - An Analysis	29
5. Conclusions	44
5.1 Result Summary	44
5.2 Future Research	45
5.2.1 Data Plane Size Variations	45
5.2.2 Connectivity Architectures	46
A. The Research Queueing Package(RESQ)	48
B. The Research Parallel Processor Prototype(RP3)	50
C. Split-Step Computation Structure	52
D. I/O Systems	56
E. Striping Schemes	60
F. Definition of Constants	62
G. Sample Reply File	63
H. Phase-Shift Computation Structure	64
BIBLIOGRAPHY	68
Vita	

List of Figures

2.1	Split-Step Migration Data Volumes	7
2.2	Split-Step Migration Computation Structure	8
2.3	Phase-Shift Migration Computation Structure	10
3.1	Logical Components of the RESQ model	12
3.2	Computation Structure for Split-Step Migration	15
3.3	Organization Schemes of Computation	18
3.4	RESQ Model of Split-Step Computation Structure	20
3.5	RESQ Model of Split-Step Computation Structure	21
3.6	Computation Structure for Phase-Shift Migration	23
3.7	I/O Subsystem for RESQ Model	24
3.8	RESQ Model for I/O Subsystem	27

Chapter 1

Introduction

1.1 Overview

A major share of research on parallel computation has been directed towards design of multiprocessor computer architectures and parallel formulation of computation-intensive applications. The result has been very significant gains in the compute capabilities realized by such multiprocessor systems. There has, however, been relatively little attention paid to parallel structuring of external memory operations and to modernizing external memory architectures.

Many scientific problems are compute bound and deficiencies in the I/O system may not become apparent. There are, however, a number of economically significant applications where very heavy demands are placed upon I/O systems. Additionally, demands for external memory operations can be expected to grow proportionally to the compute speed of the system. The problem of parallel structuring of external memory operations will thus be aggravated by multiprocessor environments.

The thrust of this thesis is the performance evaluation by simulation modeling of the execution of a family of I/O-intensive applications on a shared memory architecture. The system models include parallel structuring of I/O operations, data placement, buffer management etc., all in the context of a complete parallel formulation of the I/O-intensive application. The rest

of this chapter details the problem definition, outlines the approach, and presents a summary of the results.

1.2 Problem Statement

The problem studied in this thesis is the performance of a complete parallel formulation of a family of the 3D migration algorithms for seismic data processing. The formulation includes not only parallel formulations of the computational algorithms, but also parallel formulations of external memory operations.

The principal architectural target for the execution of this parallel computation structure is the IBM Research Parallel Processor - RP3 [1,2,3]. We have also considered shared-memory multiprocessor architectures with much faster processors to provide comparisons with current-day production multiprocessors.

3D migration algorithms are a standard element of seismic data processing and thus oil exploration. There is a wide spectrum of 3D migration algorithms. In each case, the size of the data sets are of the order of several gigabytes, necessitating large amounts of I/O activity. The computational requirements vary from being extremely computation-intensive to only moderately computation-intensive. It is often the case on conventional uniprocessors that the many passes of the data through the system required by constrained memory capacity forces even the computation-intensive algorithms to become I/O bound.

The structure of the computation is very regular. The data access patterns are quite predictable. Therefore, it is possible to construct regular and effective parallel structures for the external memory operations. It is the

case, however, that the external memory operations do include serialization points where different computational processes compete to update a shared variable.

The major thread of research has been the development of parameterized simulation models for a complete parallel formulation of a family of 3D migration algorithms. This model includes the mutual exclusion effects, architectural characteristics, resource allocation algorithms etc.

The effort in this thesis is one of the earliest attempts at viewing large-scale I/O activities in highly parallel machines. The details of the parallel formulation from the viewpoint of the geophysical application domain can be found in the References[4].

1.3 Approach

Simulation modeling has been used as a vehicle for performance evaluation in this thesis effort. The rationale behind the use of simulation is to avoid the expense of making a number of runs to explore the optimal parallel structuring from a broad set of parameters. The queueing theoretical package RESQ has been used to develop detailed models of 2 migration algorithms on the RP3.

The models propose a pipeline structure for the stream-oriented processing. The algorithm's regularity has been exploited in the data storage to improve cache-hit ratios and also to pre-fetch data sets, thus overlapping I/O transfers with processor activity. The development of highly parameterized simulation models with selective increase in resolution at points of interest has allowed for extensive simulation experimentation.

1.4 Results

Simulation runs from the RESQ models yielded considerable insight. The principal conclusions that may be drawn from them are:

- Memory-rich environments greatly alleviate the I/O problem. 3D Migration processing which is I/O bound in conventional uniprocessors is probably compute-bound in systems like RP3.
- Data organizations for highly structured data access patterns would tend to yield near linear speed-ups for parallel data transfers.
- The regularity of the data processing allows for a data set that has been read into memory to be used a number of times by different processors. Thus each processor enjoys an effective memory size far greater than that of a single processor.

1.5 Organization

An overview of seismic data processing is given in the next chapter. Chapter 3 describes the RESQ models for the migration processing. Experiments and their results are discussed in Chapter 4. We end with a few concluding remarks summarizing the research and directions for future work. Appendix A contains details about the RESQ package. Appendix B mentions the salient features of the RP3. A listing of the code for the RESQ models is presented in Appendices C through H.

Chapter 2

Seismic Data Processing

2.1 Overview of Migration Processing

Seismic data processing has been chosen as the application to drive the simulation models. The primary reasons for such a choice are the tremendous number of computational steps and the large-scale I/O transfers of big data sets. This application also evokes commercial interest, particularly in petroleum exploration. The class of seismic methods that has been selected is Migration, which refers to imaging methods employing a one-way acoustic wave equation and the ‘exploding-reflector’ model of seismic data. Two of the migration algorithms - the Split-Step Fourier and the Phase-Shift have been incorporated in the RESQ simulation models. The rest of this chapter is a discussion of both the algorithms restricted to the details pertinent to the thesis.

2.2 Split-Step Fourier Migration

Split-Step Fourier Migration is one of the more accurate migration algorithms. The ‘Split-Step’ refers to the phase-shifts being performed separately - once in the frequency domain before the inverse FFT and then in the time domain. The main disadvantage in this algorithm is the large number of computations for the 2D FFTs that makes it expensive for execution on conventional uniprocessors.

There are 2 input data volumes(3D arrays) which represent the seismic data and the velocity volume as sets of frequency(ω) planes and velocity(v) planes respectively. ω planes are complex and so each of them occupies twice the memory of a v plane which is real. NKX and NKY represent the dimensions of a single plane. A plane dimension of $1024 * 1024$ has been considered for most experimental cases. Actual plane sizes are usually smaller. 512(NF) ω planes and 1024(NZ) v planes were similarly chosen for most of the simulation runs. These numbers are typical. Figure 2.1 shows these data volumes.

Each ω plane goes through several stages in the migration processing for every depth step. Each depth step corresponds to a v plane. First, a 2D FFT is performed, followed by a phase-shift(ϕ_1) and then by the inverse FFT(FFT^{-1}). Subsequently, there is a second phase-shift(ϕ_2) for which a velocity plane is required. This yields a partial result plane(p) which needs to be accumulated to the partial results from other ω planes corresponding to that depth step. This accumulation is the evaluation of a single point of a discrete Fourier transform in the vertical direction. The accumulation of all the p planes, which are real, is also the last stage of data processing at that depth step for the ω plane. Mutual exclusive access to the partial result buffer is necessary as the partial result - a shared variable - is updated. Finally, the ω plane repeats the above for each depth step. This process is carried out for all the ω planes to yield the data volume comprising of the p planes. Figure 2.2 shows the computation structure for this algorithm.

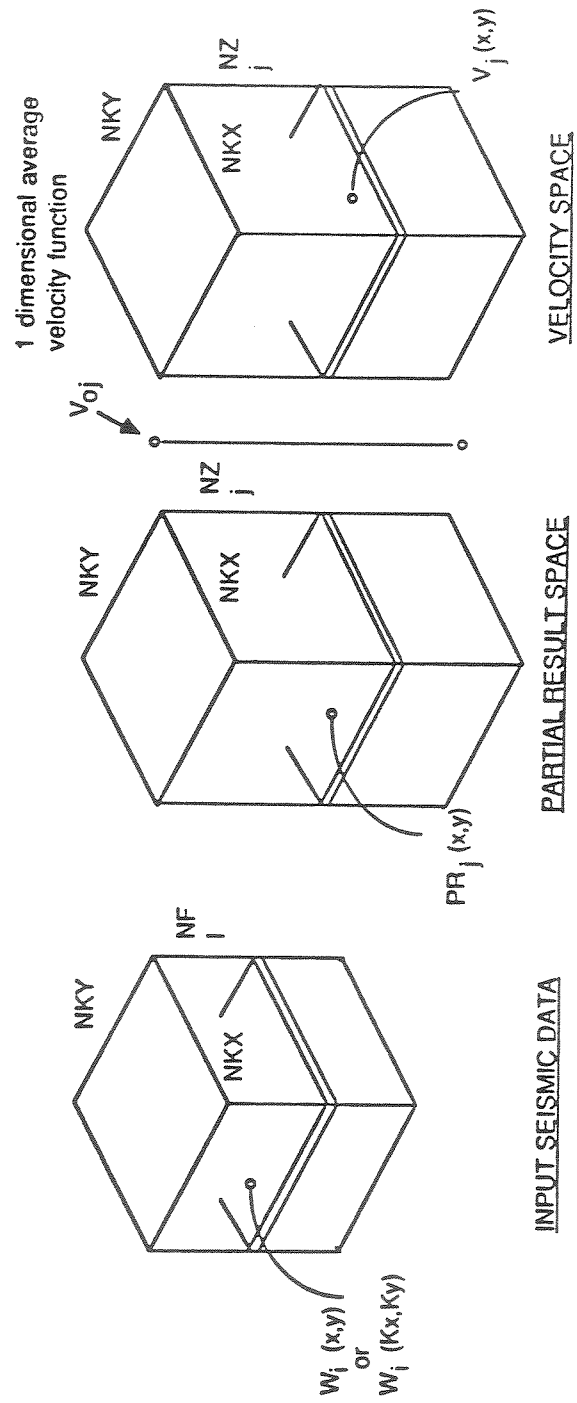


Figure 2.1: Split-Step Migration Data Volumes

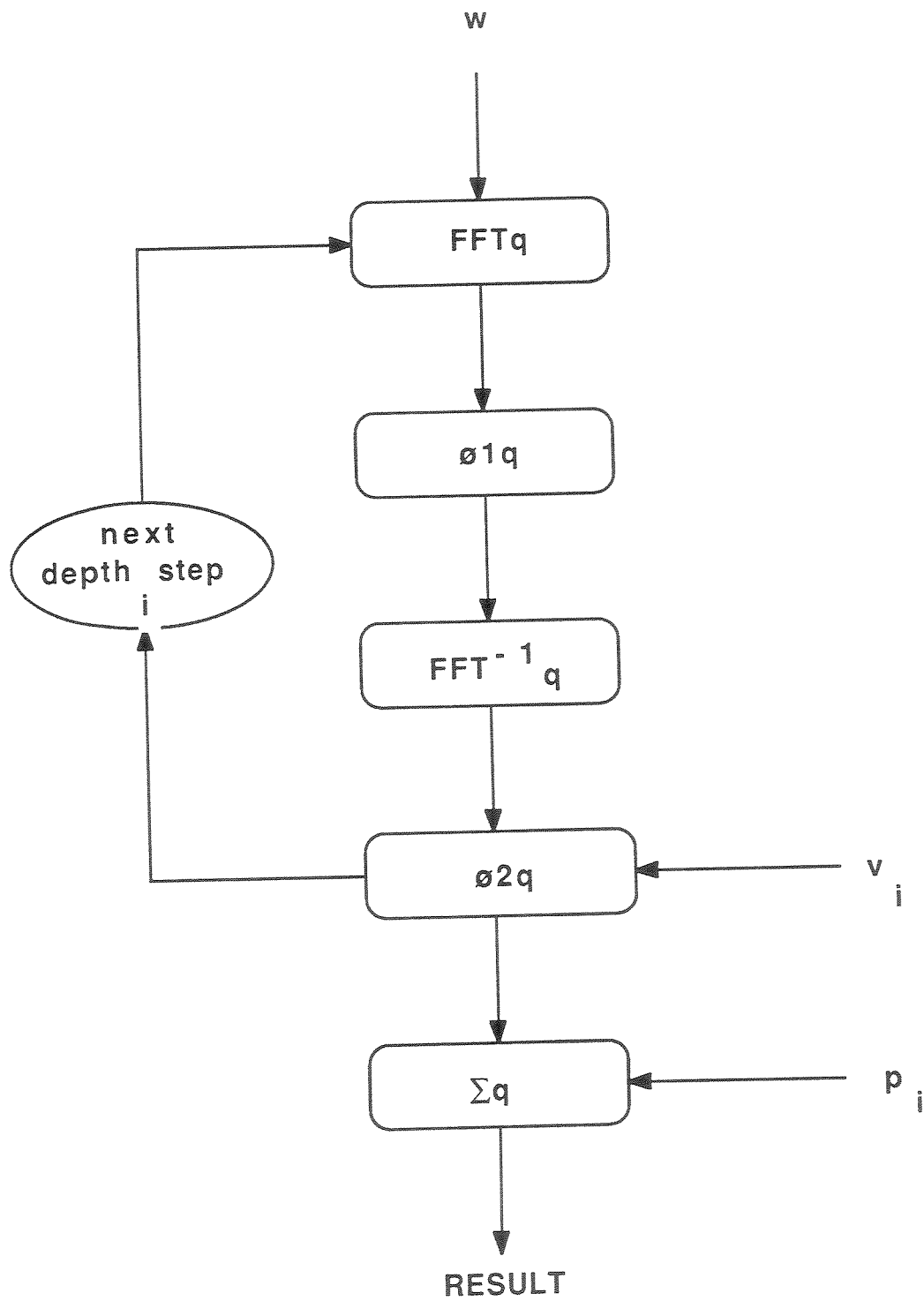


Figure 2.2: Split-Step Migration Computation Structure

2.3 Phase-Shift Migration

This is greatly similar to the split-step migration discussed in the previous section. Here, there is only one phase-shift which is performed in the frequency domain. Also, instead of using a velocity plane (representing velocities of different points at a depth step, as in the split-step migration), the phase-shift migration uses a single velocity value for all the points at a given depth. So there are no v planes in this algorithm. This makes it less accurate than the split-step but the computational expense of this algorithm is also far less. The reason for the latter is that the 2D FFTs are not present for every depth step; only the phase-shift and accumulation of partial results are. The FFTs are performed only twice - once before the data processing begins and then after all the data processing has been completed and the p planes are available. Finally, the p planes in this migration are complex unlike the split-step. The computation structure of this algorithm is shown in Figure 2.3.

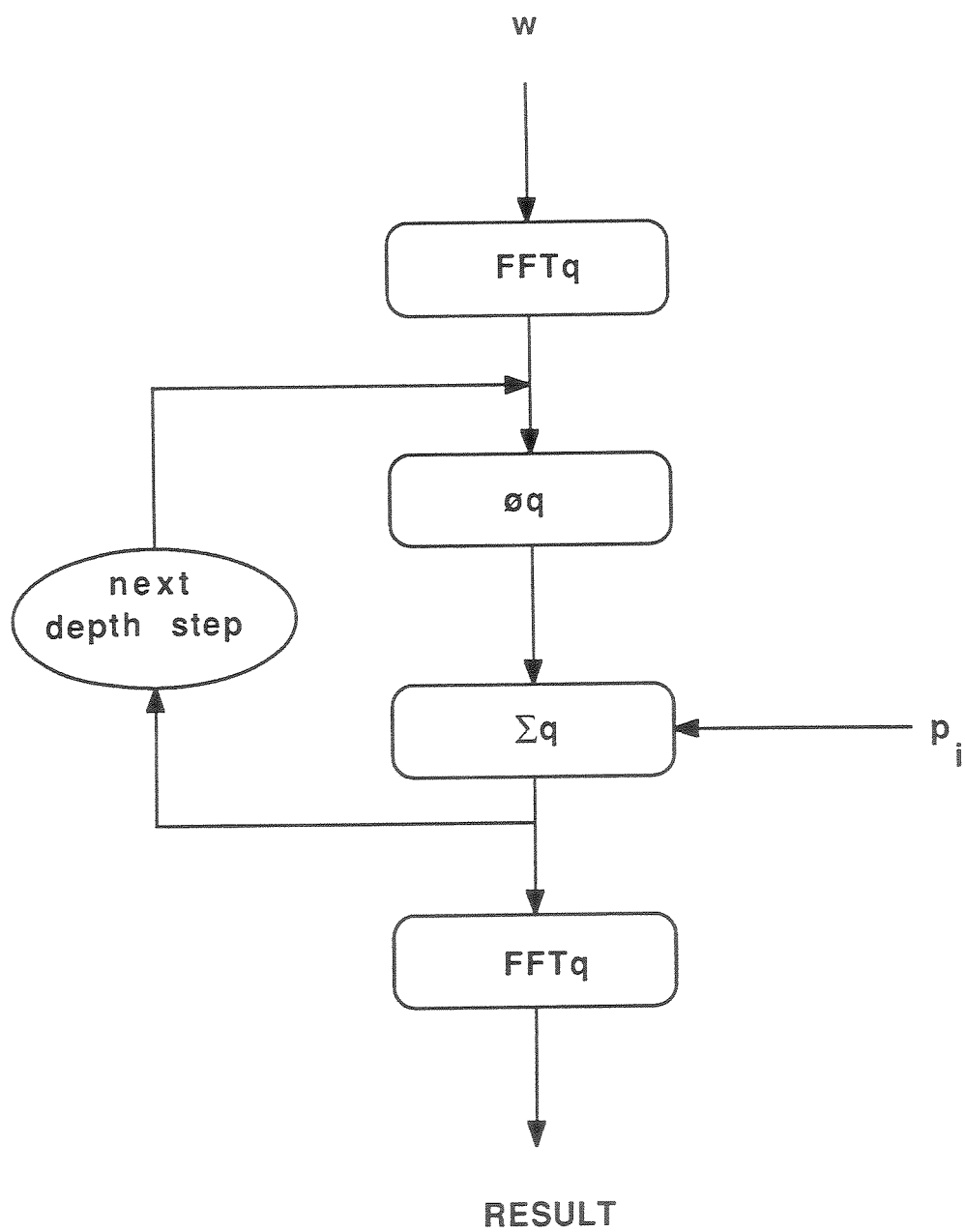


Figure 2.3: Phase-Shift Migration Computation Structure

Chapter 3

RESQ Models for 3D Migration Processing

3.1 Outline of RESQ Model

A RESQ implementation of the seismic data processing on the RP3 is discussed below. Readers are assumed to be familiar with the Research Queueing Package (RESQ)[5,6]. A brief summary of RESQ's features may be found in Appendix A. Architectural features of the RP3 are presented in Appendix B.

Figure 3.1 shows the logical components of the model. The Computation model depicts the split-step migration engine. This includes the different computation steps, processor resource allocation, data flow paths and generation of I/O requests to the Data Management unit.

The Data Management unit determines the details associated with individual I/O requests. Requests may be READ/WRITE for ω and p planes, while v plane requests are READ only. Accordingly, memory buffers are allocated and released through interaction with the Buffer Management section.

The requests are then forwarded to the Striping model which describes different disk striping schemes for data storage. Physical I/O requests are then sent to the I/O subsystem.

As a basic unit of computation in the 3D migration algorithm is a (horizontal) plane, a large portion of the RESQ model can be developed independent of the pattern of data storage on disks. Modification would

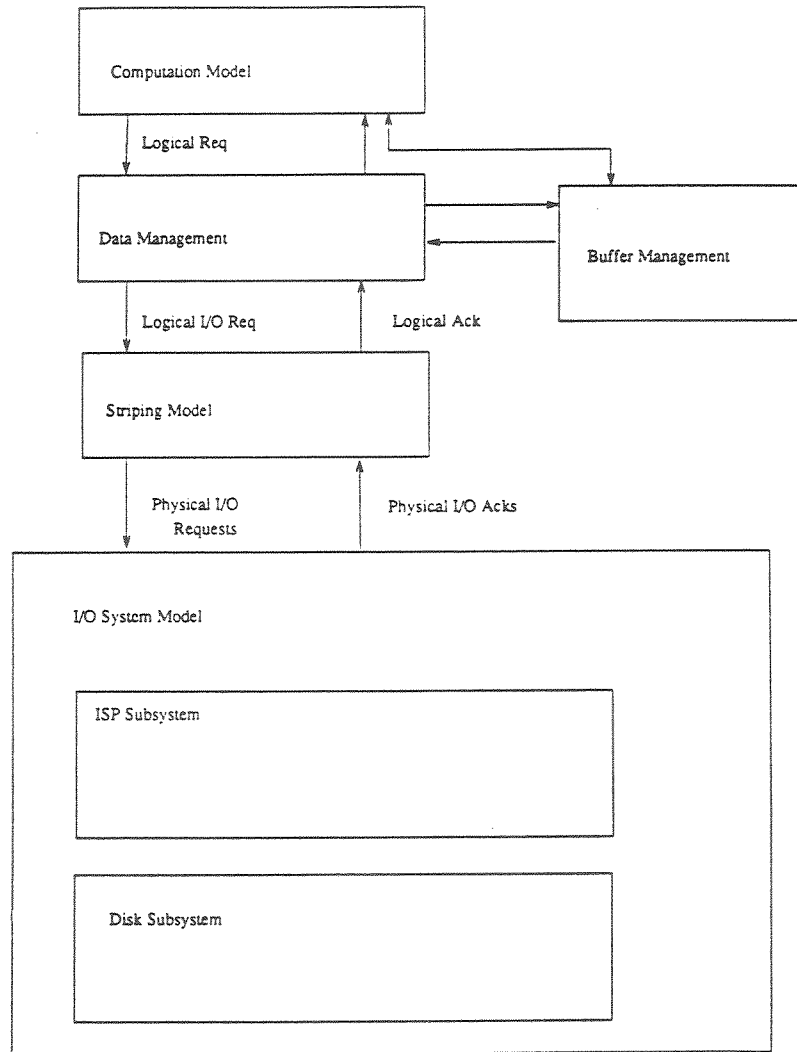


Figure 3.1: Logical Components of the RESQ model

have to be made only to the Striping sub-model for different disk striping techniques.

Section 3.2 states a few assumptions made while formulating the model. Section 3.3 discusses the Computation structure for both the migration algorithms, while the I/O subsystem is described in Section 3.4. Simulation runs and their outcome are presented in the next chapter. Appendices C through H give a listing of the RESQ code for the entire model.

3.2 Assumptions

A small number of reasonable assumptions were made while developing the model which are:

- A 64-processor RP3 with 4MB memory per processor. Other configurations had also been considered.
- All RP3 memory is configured as shared memory.
- 3380 disks with disk caches.
- High cache hit ratio owing to the sequential data access pattern.
- Hot spots caused by simultaneous access for v and p planes are negligible.
- Utilization of ISPs is low.
- Use of an ‘in-house’ FFT algorithm which does not require scratchpad memory.

The assumptions are significant, but realistic. There is a fairly substantial overhead to access data from another 64-processor block compared to another processor within the same block. Changes to the model would have

to be made to incorporate this overhead if the model is scaled up to more than 64 processors. The results presented for RP3 configurations more than 64 processors in Section 4.1 have *not* included these changes. As the data processing is extremely regular, the I/O requests are generated in a predictable fashion. This was exploited by suitably arranging the planes on disks to yield high cache hit ratios. The presence of the combining network was assumed to alleviate the hot-spot problem associated with the contention for v and p planes. The ISPs had not been modeled in very great detail as their expected use is low. The page buffers, however, had been incorporated in the model. The low ISP utilizations were later confirmed with simulation runs. Initial use of FFT algorithms requiring extra memory space suggested that this additional memory requirement would make memory buffers a critical resource. The use of an ‘in-house’ FFT algorithm eliminates memory as being a critical factor.

3.3 Computation Structure

3.3.1 Split-Step Migration Algorithm

Figure 3.2 illustrates the structure of the RESQ model describing the computation for the split-step migration.

A job $J(\omega, i)$ is an ω plane that requires data processing. Associated with each job are a number of variables of which the most important are ω and i . ω represents the ω plane associated with the job and i the iteration number. i , therefore, indicates the v and p planes required by the job $J(\omega, i)$.

Jobs are allocated job tokens from a job pool (a passive queue) based on a pre-defined priority scheme. This token serves as an access right for the job to enter the system and undergo data processing. Once a token has been

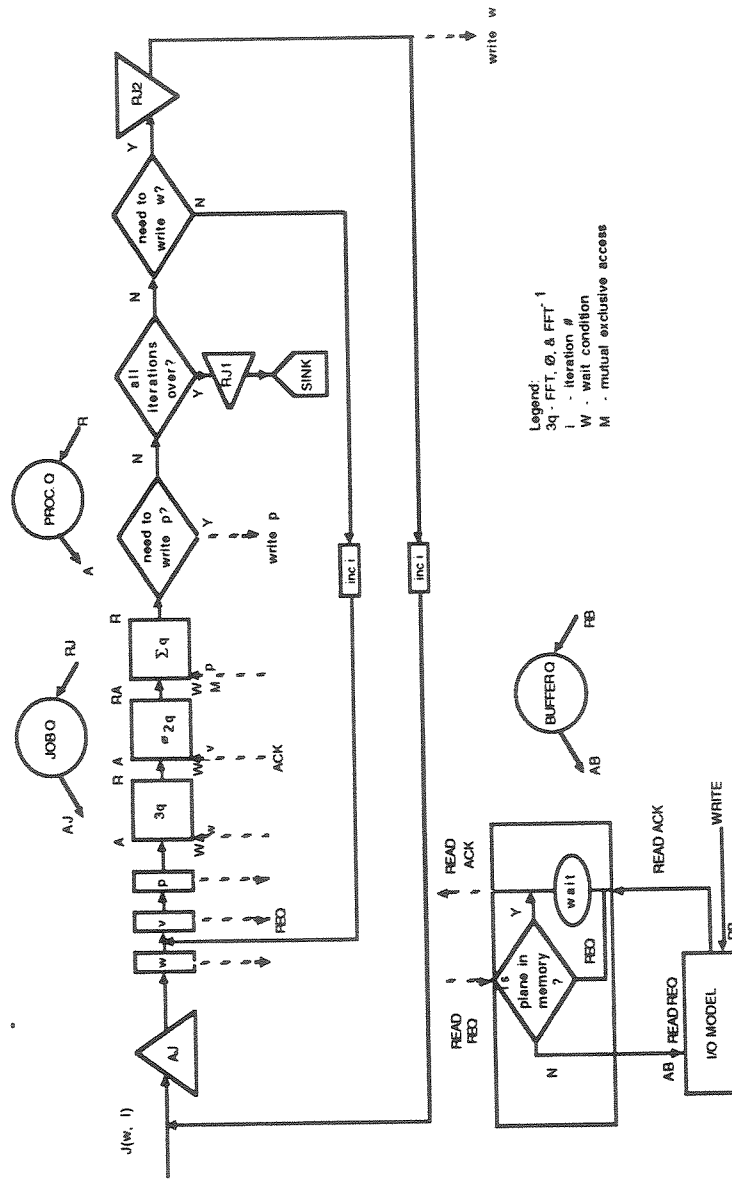


Figure 3.2: Computation Structure for Split-Step Migration

allotted, a job is allowed a specific number of iterations in the system. The number of tokens in the job pool regulates the total number of jobs in the system at any time.

A set of symbolic notations are used in Figure 3.2 to simplify the illustration. I/O requests are indicated by broken lines. A job issuing a request may be required to wait for the necessary data transfer to be completed. This wait is symbolized by 'W' before the appropriate queues. 'M' represents the mutually exclusive access for the p planes while A-R, AB-RB and AJ-RJ1, RJ2 denote Allocate-Release nodes for the processor, memory buffer and job tokens respectively.

A job entering a system needs ω , v and p planes. The ω planes are required at each stage of the computation, while the v and p planes are needed for the second phase shift and summation respectively, i.e, at the $\phi 2q$ and the $\sum q$. I/O requests for the planes can be made in the appropriate order. Further, since the requirement of v and p planes is known in advance from the value of the iteration number i , they may be requested sufficiently early. This would allow the I/O transfer of these planes to overlap with the computation involved in $3q$ (an FFT, a phase shift and an FFT^{-1}).

Each stage of the computation - $3q$, $\phi 2q$ and $\sum q$ - requires processor resource, which is modeled as a passive queue. Processor tokens are allocated and released at each of the stages yielding a dynamic allocation of processors to tasks. Different queues would receive processor resource based on factors like queue length, service time etc. This represented a flexible mapping without tying processors to particular tasks. More rigid mappings could also be represented. Use of parallel FFT algorithms, where more than 1 processor can act on a single job to provide improved performance was also allowed in

the model. This was realized by multiple allocation of processor tokens for the queues involving the FFTs, with appropriately reduced service times.

After a job had been serviced by $3q$, $\phi 2q$ and $\sum q$ (i.e, after a set of computations comprising an FFT, $\phi 1$, FFT^{-1} , $\phi 2$ and \sum was complete), it may have needed to go through additional iterations. In such a case, the job had to issue requests for the next v and p planes. A request for an ω plane was not necessary as the plane was already in memory. The need for additional iterations was determined by the organization of the computation ‘Vertical’ or ‘Horizontal’. Figure 3.3 discusses the 2 schemes in detail. Depending on the scheme employed, p planes or ω planes were written out onto the disks to free memory buffers. When all the iterations for a job had been completed, the job token was released and the job rejoined the queue for allocation of a fresh job token. A suitable priority was specified for allocation of the job tokens to ensure that the computation proceeded correctly. A check was also made at an appropriate point to see whether all the iterations (1024) had been completed in which case the job should suitably exit.

All of RP3’s memory was modeled as ‘shared’ and it was assumed that there is no overhead in accessing the memory of another processor. Therefore, there was a large single address space (minus the requirements for an Operating system and the like) available for the seismic data processing. This available memory was partitioned as buffer slots each of which was the size of a v plane. This was also the size of a p plane. However, each ω plane being complex occupied twice the memory of a v or p plane and needed 2 buffer slots. Memory had been modeled as a passive queue with each token representing the basic unit of memory - a buffer slot.

When an I/O READ request was issued, a check was first performed

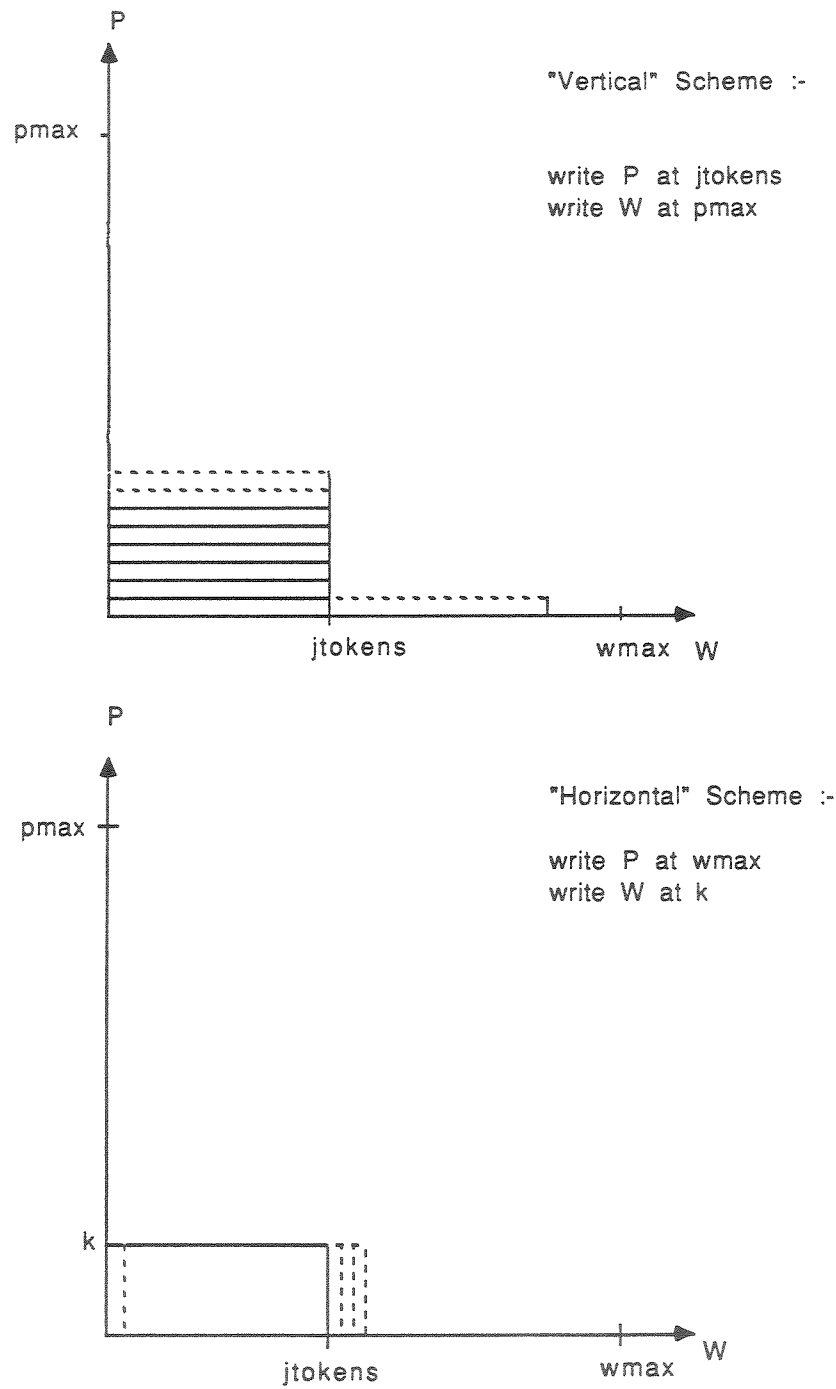


Figure 3.3: Organization Schemes of Computation

to ascertain whether the plane being requested was already in memory. If so, an acknowledgment was sent immediately. If the plane was not in memory, buffer space was first allocated for the plane (2 for ω and 1 each for v and p planes) and then the request was forwarded to the I/O subsystem. It was also possible that the plane had already been requested by a previous job, but the data transfer had not been finished. In such a case, all requests except the first were deferred till the first one had been serviced. Acknowledgments were then sent for all the requests made for the same plane. This avoided duplication of requests, needless I/O activity and wastage of buffer space. It had been assumed that the memory buffer manager was sophisticated enough to allocate contiguous buffer slots and that it was also able to effectively coalesce discontinuous empty slots. This was significant for vacancies caused by release of more than one v or p plane buffers was available for ω planes, even if the memory buffers had not adjacent.

Figure 3.4 and Figure 3.5 show the entire model using RESQ symbols for this algorithm.

3.3.2 Phase-Shift Migration Algorithm

In addition to the split-step migration algorithm (described preceding), the phase-shift migration algorithm was also modeled using RESQ. This was done to study the balance between computation and the I/O data transfers for a less computation-intensive seismic data processing algorithm.

Figure 3.6 shows the RESQ structure illustrating the phase-shift migration. As mentioned earlier in Chapter 2, this is similar to the split-step algorithm except that the two-dimensional FFT is performed only twice for every ω plane. This significantly reduces the computation expense. Also,

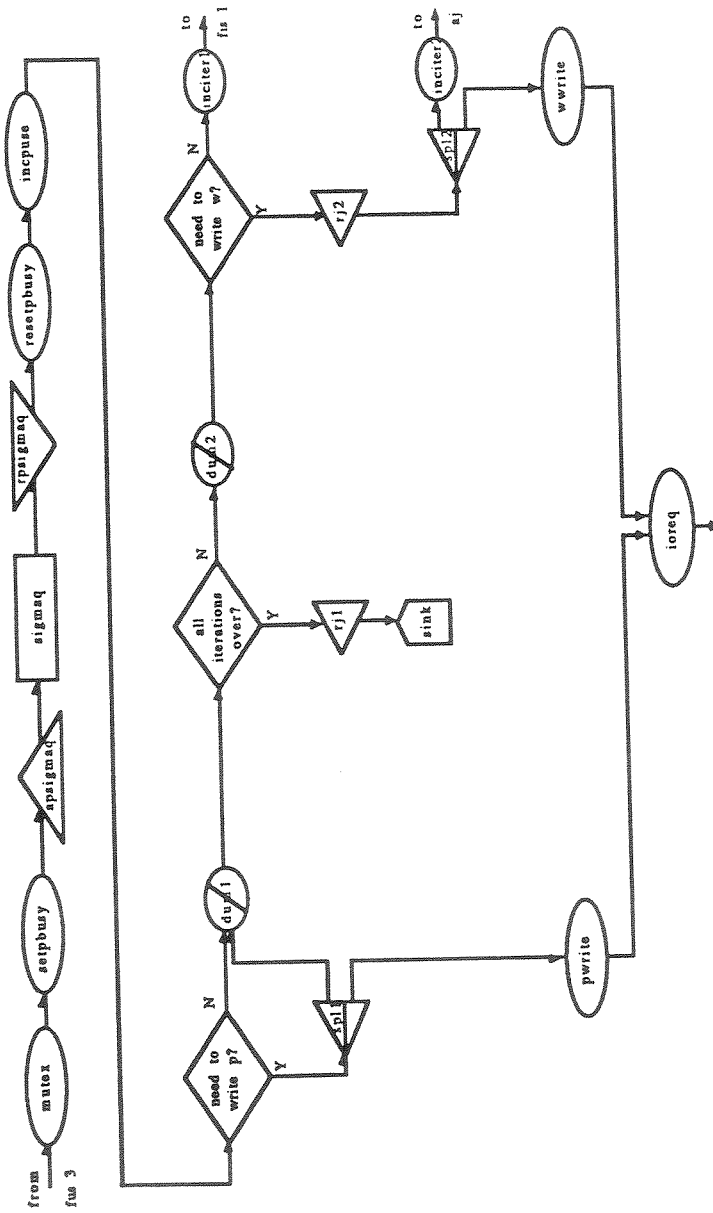


Figure 3.5: RESQ Model of Split-Step Computation Structure

there are no v planes in this migration structure and the p planes-being complex-would occupy two buffer slots.

The RESQ model for the split-step migration was modified suitably to simulate the phase-shift migration. The model would, therefore, not run very efficiently. However, the results would be accurate.

3.4 I/O System Model

This section describes the modeling of the disk striping techniques and the I/O system. The use of sub-models (allowed by RESQ) permitted different disk striping layouts while retaining the rest of the model. Figure 3.7 shows a flowchart illustrating the I/O subsystem based on which the RESQ model had been developed. The RESQ model encompassed, among others, the Page buffers, the Storage Director, the cache, the channel and the disks.

The disk striping model implemented one of the striping schemes of interest. Striping spreads the data of a single logical request (for a plane) across multiple disk heads. Multiple disks were accessed through multiple ISPs to distribute the channel and ISP loads across the system. One striping scheme gives the mapping:

$$(plane, index) \rightarrow \{ISP, CU, disk, logical\ block\}$$

Each plane was spread across a set of control units and disks and was accessed through multiple ISPs. (If all accesses were directed through a single ISP, which is possible in the initial RP3 I/O configuration, much of the benefit of striping would be lost since the data transfers would be contending for a single channel).

Two different striping schemes had been modeled. In the base case

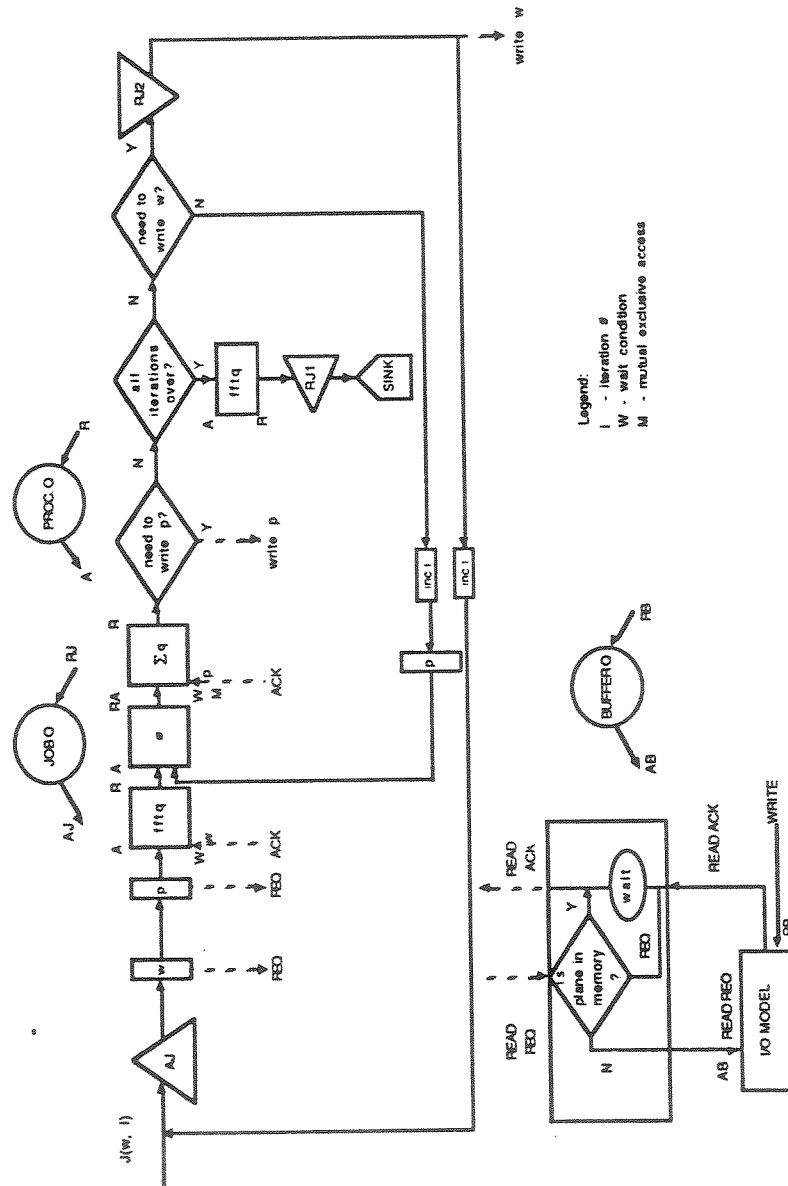


Figure 3.6: Computation Structure for Phase-Shift Migration

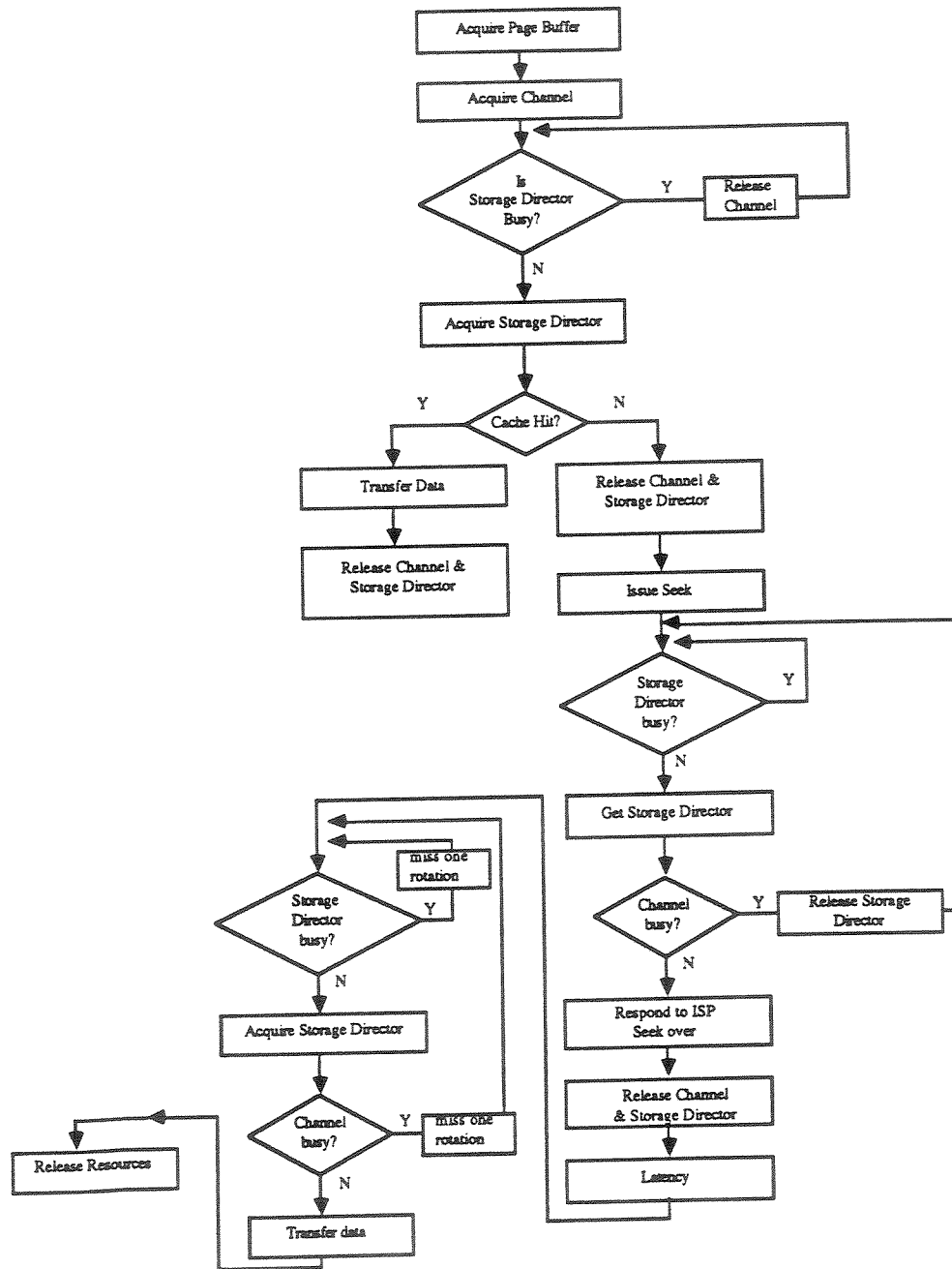


Figure 3.7: I/O Subsystem for RESQ Model

where no striping was employed, all the data for a set of planes was stored on a single disk. Different sets were stored across different control units. In the case where striping was utilized, each plane was spread evenly across a single disk on each control unit, i.e., the data for a single plane resided across eight different disks of eight different control units.

For an I/O transfer, it was necessary to obtain simultaneous possession of several resources. The page buffer had to be acquired initially. Then both the channel and the Storage Director were required. After the channel had been acquired, an attempt to possess the Storage Director had to be made. If the Storage Director was busy, then the channel was released to allow for its possible use to service other I/O requests.

Once the channel and Storage Director had been acquired and in case of a cache-hit, data transfer commenced. After data transfer, the channel, the Storage Director and the Page buffer were released. As the pattern of I/O requests was regular and known in advance, the data was stored on disks in an appropriate manner to improve the cache hit ratio. In fact, a proper data storage technique would result in a cache hit for all READ requests except for the first set of planes.

In case of a cache miss, both the channel and the Storage Director were released. The disks had also been modeled as a passive queue. On acquiring the disk, a 'seek' is issued after which the channel and Storage Director had to be simultaneously acquired again. A response was sent to the ISP indicating the completion of the 'seek' after which the channel and storage Director was once again released. There is a latency involved in disk access which for simulation purposes was assumed to be constant. A 'seek' has been modeled to take 16ms and the latency 8ms. After the latency delay,

both the Storage Director and the channel had to be acquired. If either was busy a rotation was missed, until both could be simultaneously obtained. Once both were in possession, data transfer was performed, completing the service of the I/O request, after which all resources were released. However, the RESQ model had not been developed to this detail as a cache miss would be rare for reasons discussed before. Figure 3.8 shows the extent to which the model had been implemented.

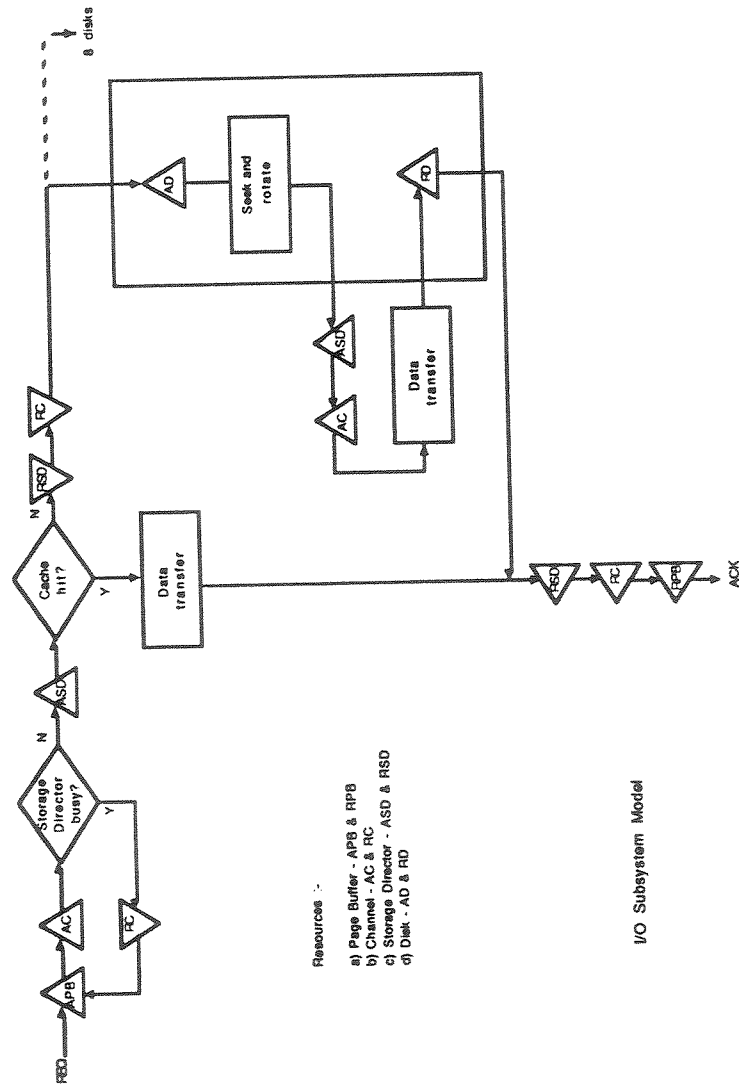


Figure 3.8: RESQ Model for I/O Subsystem

Chapter 4

Experiments and Results

4.1 Simulation Experiments

The parameterized nature of the RESQ model facilitates extensive experimentation. Some of the more interesting variations for simulation runs are enumerated below:

- Number of processors and their performance
 1. (801-like) RP3 processors
 2. Faster processors (2x 4x 8x 10x)
 3. Eight or sixteen 3090-like processors
- Plane sizes and number of planes in a data space (base case is 512 x 512 x 1024).
- Degree of parallelism within the processing of a single plane.
- Allocation priorities for processors and memory buffers.
- Disk striping schemes.
- Organization of the computation ‘Vertical’ and ‘Horizontal’.
- Memory capacity per processor.
- Flexible and rigid mappings of processors to tasks.

4.2 Results from Simulation Models

The Split-step and Phase-shift formulations of 3-D migration can be cast in almost identical parallel computation structures. The primary difference is that the Phase-shift algorithm requires only two FFTs to be executed during the processing of an ω plane against all velocity planes while the Split- Step requires an FFT for each ω and v plane pair. Since the FFTs dominate the total computation time the result is that the Phase-Shift algorithm has a very much lower amount of computation per data element. Thus use of these two methods yields stress conditions for the computation portion of the architecture and a stress condition for the I/O architecture. The results of the simulation model evaluations are given in Tables 4.n ($n = 1 - 5$) and in Graphs 4.n ($n = 1 - 6$).

Each table summarizes the resource utilization for a parameter set. Graphs 4.1, 4.2 and 4.3 (Split-Step) and 4.4, 4.5 and 4.6 Phase-Shift) give the results for 64 processor, 8 ISP RP3-like configuration for processor speeds from 1 MFLOP to 50 MFLOPs for striped data organizations. The heavy compute requirements of the Split-step algorithm lead to compute dominated resource utilizations until processor speeds of up to 50 MFLOPS are used. The lower compute requirements of the Phase-shift algorithm lead to an I/O bottlenecked system for 10 MFLOP processors.

4.3 Simulation Runs - An Analysis

Simulation runs were performed on the RESQ model with parameter sets that appeared to be of greater interest. Those experimental cases were:

- Number of processors - 64-processor and 512-processor RP3 configurations were considered.
- Processor speeds - In addition to the base case of a 1-MFLOP processor, 5,10,30 and 50- MFLOP machines were also included.
- Degree of parallelism or Concurrency factor - The parallel processing for the FFT within a single plane was simulated for a concurrency of 1,8,10, 12 and 16.
- Disk striping - Simulations with and without disk striping were performed. Results discussed below are for cases employing disk striping. Experiments without disk striping did not offer any advantages and evoked little interest.
- Organization of the computation - The ‘Vertical’ scheme was used in all experiments. However, the number of interactions with ω planes after which a p plane was written out onto disks was altered to study its effect.

Simulations were performed with both the Split-Step and Phase-Shift migration models.

All the simulation runs were for a data space size of 512x512x1024. The number of ω planes was 512 and the number of p planes was 1024. These represent typical problem sizes. Allocation of processors and memory buffers to jobs were on a first-come-first-served basis, as long as the computation flow required by the ‘Vertical’ scheme was satisfied. A flexible mapping of processors to tasks had been used throughout. Each processor had been modeled with a memory capacity of 4 MB all of which was considered ‘shared’.

Service times for the FFT, the phase-shift and the addition of partial products were obtained by executing sequential FORTRAN code on a Sequent Balance. These times were scaled to reflect corresponding times on a RP3-like processor and are tabulated here.

Split-Step Migration - Service times

Queue	Service time(sec)
$3q$	67.7
$\phi 2q$	6.2
$\sum q$	0.14

Phase-Shift Migration - Service times

Queue	Service time(sec)
FFT _q	31.7
ϕq	6.2
$\sum q$	0.14

The concurrency within the processing of a single plane (using parallel FFT algorithms) had been assumed to be ideal. If the FFT time for a sequential algorithm is 't', then the corresponding time for a parallel implementation with 'p' processors was assumed to be 't/p'.

The 'Vertical' scheme for computation flow stipulates the following 2 conditions:

1. The number of jobs allowed in the system should equal the number of ω planes that have interacted with a particular p plane, before the latter is written onto the disks.
2. Each ω plane is written out only after it has undergone processing with all p planes.

These conditions correspond to the maximum possible use of a data plane in memory before it was transferred to the disks.

The simulations were performed in a planned fashion, with the features of interest from earlier runs suggesting parameter sets for the simulations to follow. The discussion below traces those experiments.

It may be observed that I/O and processor utilizations are not uniform throughout the computation. Early load on the I/O system is high as a specified number of ω planes have to be initially read into memory. There would also be I/O requests for v and p planes as the computation proceeds. This initial burst of I/O activity is seen until the said number of ω planes that are allowed in the system have been read into memory. I/O utilization then falls to a low value which reflects the channel loads that correspond to transfers of v and p planes as the data processing progresses. The only increase in I/O activity thereafter is for writing out ω or p planes onto the disks. However, this increase will not reach the high initial utilizations as at this stage, the computation will be skewed sufficiently such that the earliest job is well ahead of the last. The simulations were not run long enough to reach this stage of computation.

The utilization pattern for processors is different. Initial utilizations are low as processors wait for I/O requests to be serviced. As more

data planes reach memory, processor utilization increases as computations for these planes begin immediately. Processor utilization reaches a peak when a sufficient number of data planes have reached memory to keep all or most of the processors busy.

The preceding discussion suggests that the point at which statistics are gathered is crucial. There are specifically 3 time intervals each of which yield statistics significantly different from each other. The 3 intervals correspond to an initial phase when I/O utilizations are high and processor utilizations are rising, the final phase - a steady state - when I/O utilizations are low and processor utilizations are high and a transition phase when the I/O utilization curve reaches the 'knee' and processor utilization levels off to a maximum value.

Data from the simulation runs were gathered from all the 3 phases, particularly before and after the final phase began. Graphs 1 to 6 show processor and channel utilization percentage values of the first two phases and reflect corresponding values for the final phase gathered independently, ie, the utilizations corresponding to the first 2 phases were discarded and data acquisition was restarted for the final phase alone. The results show utilization values which closely match the algorithm's pattern.

The initial set of runs were performed for the Split-Step migration algorithm. A 64-processor configuration was chosen for the first set of experiments. Graphs 4.1,4.2 and 4.3 show utilization values for 3 different processor speeds(1,5 and 10 MFLOP), with disk striping. Note that the simulation time axis is not linear. It is seen that with increased processor speeds, I/O work-load increases as data sets are requested more frequently. This increased channel utilization would force processors to wait for I/O requests

to be serviced, sharply reducing processor utilization. The runs suggest that this migration algorithm would achieve a good computation-I/O balance for a processor running at about 10 MFLOPs- where both processor and channel utilizations are reasonable.

The same test case was repeated for the Phase-Shift migration. Results are displayed in Graphs 4.4 to 4.6. As in this algorithm, the FFTs are performed only twice (compared to 1024 in the Split-Step algorithm) for each ω plane, channel saturation is seen even at low speeds. It may be inferred that high processor speeds would be unnecessary for the Phase-Shift algorithm and a fair balance between computation and I/O is achieved at speeds close to or a little higher than 1 MFLOP. This is significantly less than the corresponding speed of 10 MFLOP for the Split-Step migration.

Next, a 512-configuration RP3 was considered for the Split-Step migration. However, the data planes were allowed to reside on disks across one group of 8 ISPs. The RP3 provides 8 groups of 8 ISPs each, with each 8-ISP group for a 64-processor block. 64 jobs were allowed within the system at any time. A concurrency factor of 8 was chosen for the FFTs. In the RESQ models, the service times of $3q$ was changed to an eighth of the sequential uniprocessor timing to reflect this parallelism. Table 4.1 illustrates the steady-state results for 3 different processor speeds. Again, at high processor speeds, the jobs are forced to wait for I/O service pushing up channel use and bringing down the use of processors. But even at 5-MFLOP speed, both processor and channel utilizations are relatively low. This may be explained by the fact that while a concurrency of 8 is applied on the FFTs, all the 512 processors are busy only if all the 64 jobs are in $3q$. All jobs which are not in $3q$ -those in $\phi 2q$, $\sum q$ or just waiting for some other resource - do not engage 8 processors. This

tends to decrease processor utilization. Thus, for a better use of processors, greater concurrency needs to be applied on each plane for the FFTs or the number of jobs(ω planes) within the system has to be augmented or both.

This may be summarised by the following condition:

$$(\#jobs\ in\ system) * (concurrency\ factor) > \#processors\ in\ system$$

Greater concurrency was then considered for the same migration algorithm. Results in Table 4.2 are very similar to those in Table 4.1. The increased concurrency had its effect. At 5-MFLOP speed, processor and channel utilizations were 83.6% and 10.1% respectively when the concurrency factor was further increased to 16. We note, however, that processors are not being maximally used. An explanation for this behaviour is that increased concurrency reduces the service time of $3q$ making this time comparable or even less than the service times for the other queues. Thus the jobs speed through $3q$ while taking long times in the other queues. This defeated the purpose of multiple allocation of processors to a single job at $3q$. Concurrency beyond a threshold, therefore, may not be useful. This is true only if the service times of $3q$ is allowed to fall in proportion with the increase in concurrency. But, if this assumption is valid, we conclude that high concurrencies in FFT algorithm do not yield a proportional improvement in performance, if the service times of the queues with the FFTs decrease below the other queues. Also, the RESQ model had been developed such that jobs would wait before the queues involving FFTs for allocation of all the processors (specified by the concurrency factor) before processing commences. High concurrency factors would mean longer waiting times to simultaneously acquire all processors.

After studying the effects of the concurrency factor, the number of jobs allowed inside the system was investigated. Again the split-step migra-

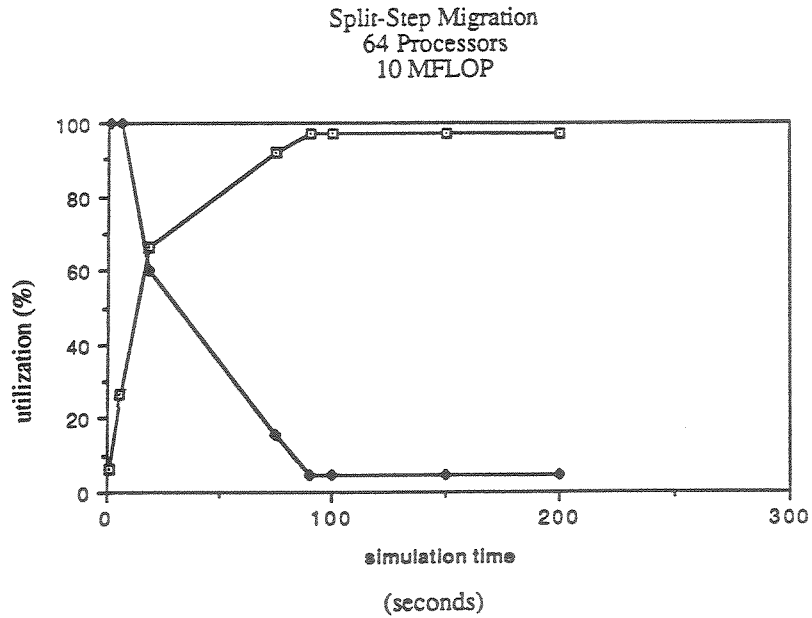
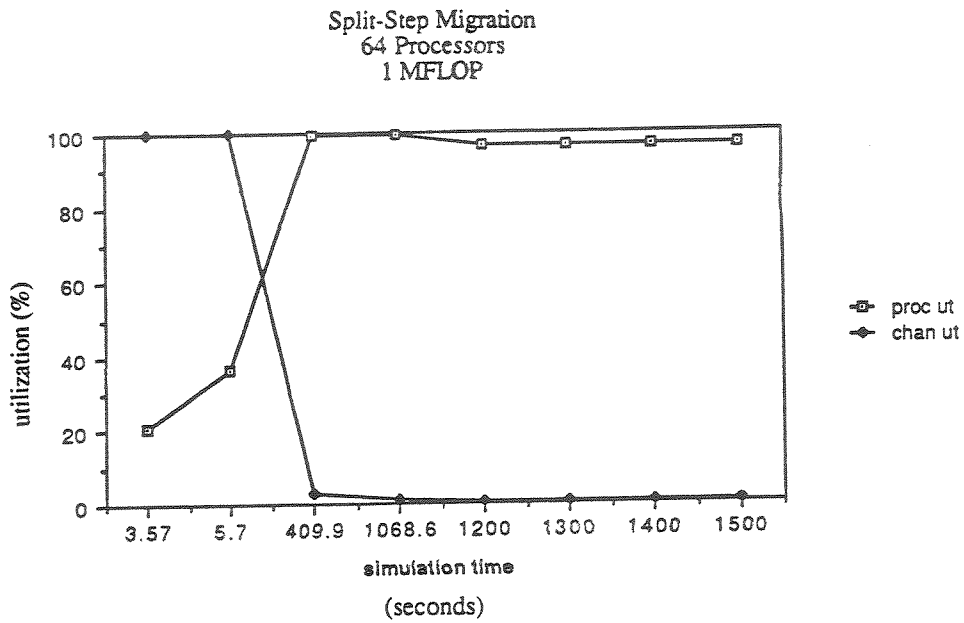
tion algorithm was chosen for the sake of comparison. The simulations were performed for a processor running at 5 MFLOPs and for a concurrency factor of 8. Table 4.3 shows the results for 3 cases. Increasing the number of job tokens from 64 to 96 and then to 128 yielded better utilization values. The channels had not been saturated.

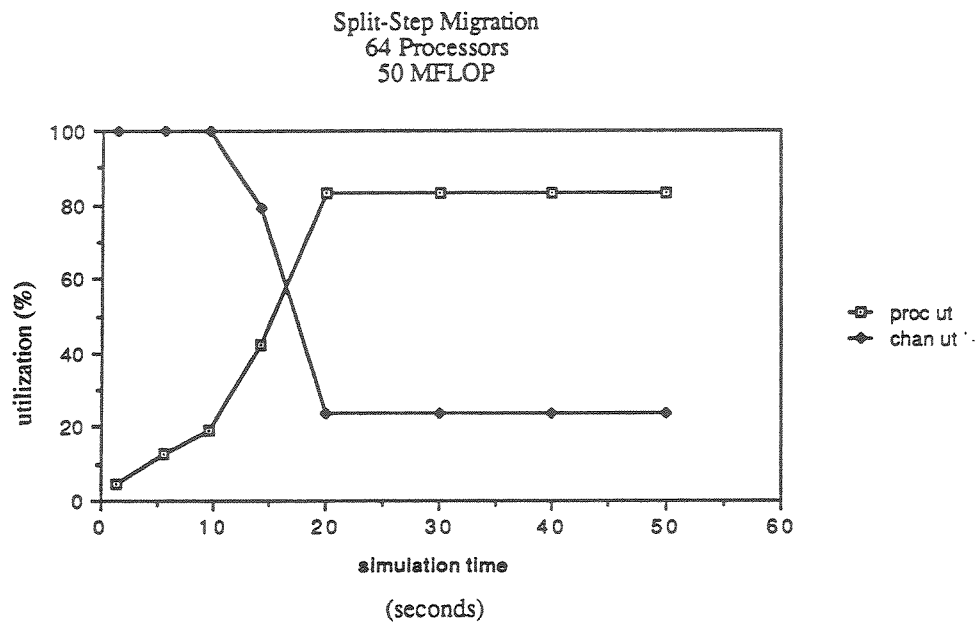
The effect of enforcing mutual exclusion for updating the partial results was then studied. This was practically realized by turning ‘off’ the mutual exclusion and comparing the results with mutual exclusion present. The effects shown in Tables 4.4 and 4.5 for both the migration algorithms were not significantly different. In most experiments, the differences in utilizations were within 2%. It is difficult to accurately determine statistical variability and error for such low deviations and, therefore, to draw inferences based on such data. However, it may be noted that no drastic change is seen by the presence or absence of mutual exclusion. The degree to which the computation among different jobs had been skewed was one of the reasons for the mutual exclusion having minimal influence on the resource utilizations. Sequential servicing of I/O requests is one of the reasons for causing this skew. The skew is less pronounced for the phase-shift migration as the FFTs are not performed repeatedly, causing the mutual exclusion effects to be greater for this case.

Both the seismic data processing algorithms have regular and predictable data access patterns. Traces of simulation runs showed that data planes were read into memory in the manner predicted by the algorithms. The simulations were not run long enough for the planes to be written onto the disks. The algorithm also predicted high initial loads on the I/O system to read in data planes, after which the I/O utilization falls. Data processing,

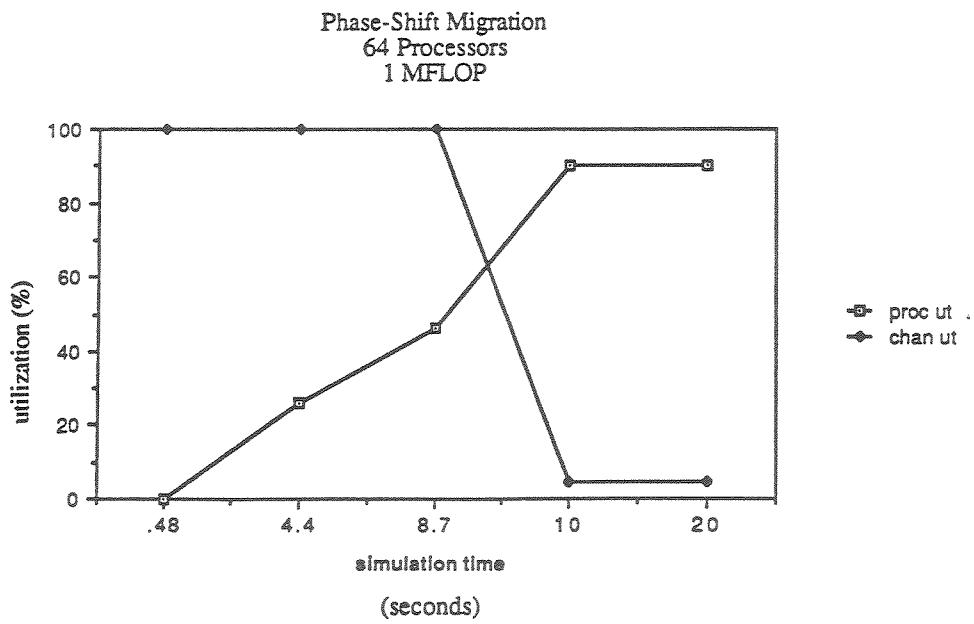
however, commences after the planes have arrived in memory and processor utilization increases thereafter. Data acquisition at different points of time during simulation runs established this.

A final note - the entire Split-Step migration algorithm computation involves each of the 512 ω planes interacting with each of the 1024 v planes for a total of 524288 such interactions. By keeping track of the number of actual interactions in simulation runs, it is seen that for most experiments, less than 1% of the total data processing was completed.

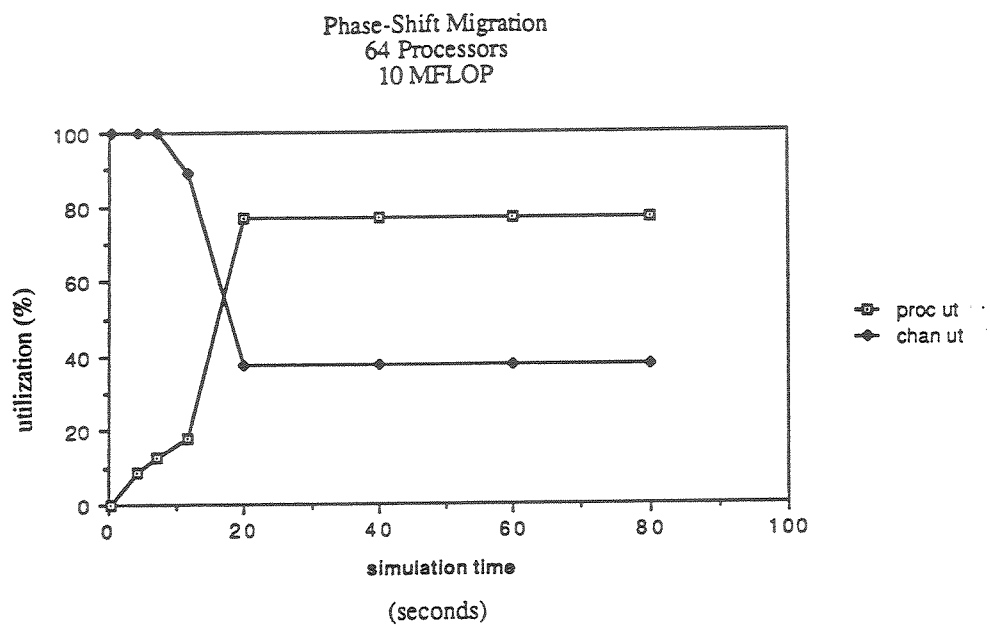




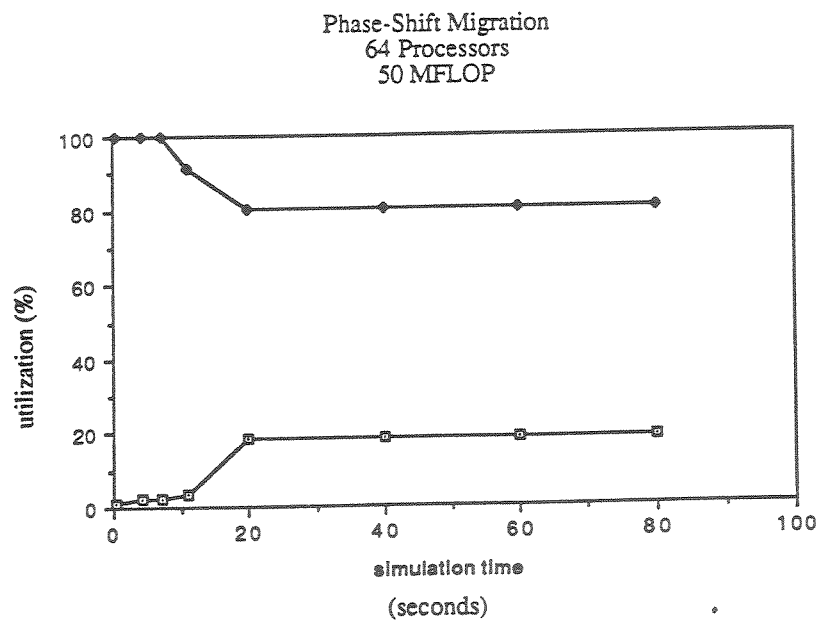
Graph 4.3



Graph 4.4



Graph 4.5



Graph 4.6

SPLIT-STEP MIGRATION
 512 PROCESSORS
 JTOKENS = 64
 CONCURRENCY FACTOR = 8
 STEADY-STATE

SPEED (MFLOPS)	UTILIZATION %	
	PROCESSOR	CHANNEL
5	61.4	7.0
10	59.3	14.2
30	39.3	59.3

TABLE 4.1

SPLIT-STEP MIGRATION
 512 PROCESSORS
 JTOKENS = 64
 STEADY-STATE

SPEED (MFLOPS)	CONCURRENCY FACTOR = 10		CONCURRENCY FACTOR = 12	
	UTILIZATION %		UTILIZATION %	
	PROCESSOR	CHANNEL	PROCESSOR	CHANNEL
5	69.0	8.0	75.5	8.8
10	66.4	16.2	72.1	17.6
30	43.0	62.2	43.3	65.4

TABLE 4.2

SPLIT-STEP MIGRATION
512 PROCESSORS
CONCURRENCY FACTOR = 8
5 MFLOP
STEADY-STATE

JTOKENS	UTILIZATION %	
	PROCESSOR	CHANNEL
64	61.4	7.0
96	86.4	13.0
128	91.5	15.7

TABLE 4.3

SPLIT-STEP MIGRATION
64 PROCESSORS
CONCURRENCY FACTOR = 1
STEADY-STATE

SPEED (MFLOPS)	ME PRESENT UTILIZATION %		ME ABSENT UTILIZATION %	
	PROCESSOR	CHANNEL	PROCESSOR	CHANNEL
1	96.8	0.4	97.0	.45
10	97.3	4.6	96.4	4.4
50	83.3	23.6	84.2	24.2

TABLE 4.4

PHASE-SHIFT MIGRATION
 64 PROCESSORS
 CONCURRENCY FACTOR = 1
 STEADY-STATE

SPEED (MFLOPS)	ME PRESENT		ME ABSENT	
	UTILIZATION % PROCESSOR	UTILIZATION % CHANNEL	UTILIZATION % PROCESSOR	UTILIZATION % CHANNEL
1	92.6	4.7	95.5	5.0
5	78.9	20.4	83.9	22.3
10	77.11	37.7	78.7	42.4

TABLE 4.5

Chapter 5

Conclusions

5.1 Result Summary

The points of significance from the simulation runs have been presented in the previous section along with the description of the experiments. They are paraphrased here:

1. High channel loads seriously affect processor utilization.
2. Disk striping alleviates saturation of the I/O system. It is particularly suited for high processor speeds when the I/O loads are greater.
3. A reasonable processor-I/O balance is obtained in the Phase-Shift migration for speeds which are significantly less than corresponding speeds for the Split-Step migration.
4. High concurrency factors are useful only as long as the service times of the FFT queues are higher than the other queues.
5. Higher number of jobs allowed in the system at any time considerably improves processor utilization without saturating the I/O system.
6. Data access patterns are very regular as predicted.
7. Preliminary simulations suggest that mutual exclusion effects may not be serious.

Seismic data processing involves tremendous amounts of computation and data transfers. The computation, particularly those involving the FFTs, are long. Data access patterns are regular and predictable. Also, multiprocessor systems like the RP3 enjoy large memory capacities. All the above factors alleviate the I/O problem. Seismic data processing, which was I/O intensive in uniprocessor systems has become quite computation-intensive in a system like the RP3.

5.2 Future Research

Though the simulation models have produced a number of interesting results, it could be further exploited to give more directions for the parallel structuring of the problem under study. The nature of implementation of the models allow for the above to be achieved with relatively minor changes to the model.

5.2.1 Data Plane Size Variations

A common problem for any experimental case arises when the data plane size increases beyond a threshold of 2 MB. This is because there are 8 Page Buffers in a 64-processor block, each with a capacity of 256 KB for a total of 2 MB. Hence a single (logical) plane request should be manifested as repeated physical requests for plane sizes greater than 2 MB.

For a plane consisting of N points ($N=n*n$, where the plane is of dimension $n * n$), data transfer time would be in the order of N . The FFT which is done for each point would be $N\log N$. So for increased plane sizes, the FFT time would increase by a factor of $\log N$ faster than the data transfer time. Thus the problem becomes more compute-bound for greater plane sizes.

5.2.2 Connectivity Architectures

It would be interesting to study the effect of parallel structuring on an architecture involving connectivity. Such a message-passing system would not be able to offer many of the advantages that ‘shared’ memory could.

Mapping: The mapping of processors and other resources to jobs and the memory-model adopted are dependent on each other. A flexible mapping like the one used so far is possible only with a shared-memory machine. For a non-shared configuration only a rigid mapping is feasible. A rigid mapping may be tried in the RP3 to squeeze out additional performance by private data access.

Flexible mapping - This has been used thus far, with a shared-memory-model. Here change in data plane sizes would affect executions in the manner described in the previous section. Also, if the data plane size increases beyond the capacity of a single processor there is no effect as all the memory is ‘pooled’. Changes in memory capacity of processors also not be affected by the model as all resources are treated as a common resource. At present, memory is not a critical resource. So the capacity of each processor could be reduced considerably before memory becomes a constraint.

Rigid mapping - This mapping may be used with message-passing machines or in a RP3 configuration aimed at maximizing performance. The gain in performance may be viewed as follows:

The overhead for private access compared with shared access in the RP3 is 1:1.60. In the flexible mapping scheme, all accesses are shared and the overhead corresponds to 1.60. In a rigid mapping scheme, ω planes would reside in private memory while v and p planes would be shared. Hence ω

plane accesses are cheaper. For modeling, the key is to have a good estimate of a fraction ' f ' of private ω plane accesses of all memory accesses. This would reduce the *average memory access time* from $[1 * 1.60]$ to $[1 * f + (1-f)*1.60]$. $f=0$ implies pure shared access, while $f=1$ indicates pure private access. Phase-Shift migration is a good vehicle for this case as v planes are absent and p plane accesses would be few compared to private ω plane accesses.

Here each job will be allocated resources which it would hold for all the queues *and* for all the iterations, ie, the initial resources allocated would be released only when the job exits the system.

Mutual Exclusion effects would be 'real' as processors and memory buffers would not be utilized during 'locked' periods. This contrasts the flexible mapping where processors would not be allocated unless p planes are free.

There are greater constraints on memory capacity and data sizes. It is imperative that ω planes fit into one processor for private access.

Also, in a rigid mapping, concurrency > 1 would be irrelevant, for the processing of a single plane.

Connectivity for External Memories: The interconnection network(along with its message-combining facility) between the processors and the memories of the RP3 is one of the contributing factors for the machine's capabilities. It would be interesting to consider some reasonable form of connectivity between the processors and the disks as well. Such connectivity may result in greater parallelism for I/O activity. This has particular significance for partitioned memory systems.

Appendix A

The Research Queueing Package(RESQ)

The Research Queueing Package(RESQ) is a special-purpose modeling language with a rich set of modeling primitives that can be used to effectively develop simulation models for analyzing resource contention systems.

Typically a RESQ model would consist of a population of jobs, sets of queues and nodes, rules to specify the movement of jobs among different elements of the model and details about the solution the model should provide.

Jobs are objects which require use of system resources. Such resources are usually represented by queues. There are 2 kinds of queues - *active* and *passive*. Active queues generally denote servers, while passive queues allow for the representation of simultaneous resource possession, blocking effects and the like. Passive queues have a pool of *tokens*, with each token representing a distinct unit of a resource. Different priority schemes for allocation of the passive queue tokens are supported.

Different nodes like *Set*(to assign values to simulation variables), *Wait*(to ensure certain boolean conditions are satisfied), *Dummy* (to help specify job movement), *Split*(to replicate a job), *Fission* (to create a child job) and *Fusion*(to merge child jobs with parents) provide powerful simulation capability. Use of *Submodels* permit development of large simulation systems in a modular fashion.

RESQ can provide performance measures by both numeric analysis and simulation. They include resource utilization, throughput, average queue length etc. RESQ also generates confidence intervals to determine accuracy of the simulations to overcome statistical variability.

Appendix B

The Research Parallel Processor Prototype(RP3)

The Research Parallel Prototype(RP3) is a highly parallel, shared-memory MIMD machine with 512 processors. It is expected to provide a peak performance of 1.3 GIPS, 2 GB of main memory, a peak I/O bandwidth of 192 MB/sec across 64 channels and inter-processor communication rates upto 13 GBytes/sec.

Processor-Memory Organization - RP3's 512 processors are organized into 8 groups with each group having 64 processors. Each processor is a 32-bit microprocessor based on the RISC philosophy. Each processor supports 4 MB of memory(of which any part may be classified 'private' or 'shared') with a 32 KB cache. A low-latency bipolar network allows a peak inter-processor communication speed of 13 GByte/sec.

All processors are linked to all the memory elements through a modified Omega or Banyan network. A *combining* network has been used to alleviate the 'hot-spot' problem which could otherwise lead to performance degradation. The interconnection network is part of the memory access path and there is a single absolute address space.

Each memory reference goes through a mapping process. If the reference is not satisfied by the 32 KB private cache, it proceeds to the network interface. If the required data is in the requesting processor's memory, then

the data may be obtained without going through the network. If, however, the data resides in a different processor's memory, the reference is sent to the target node over the network. The memory access time ratios for cache:local memory:shared memory is 1:10:16.

I/O Organization - The I/O organization of each 64-way subsystem is symmetric, but independent with respect to the other 64-way units.

Each group of 64 processors is further subdivided into 8 groups each. Every 8-processor group is supported by an I/O and Support Processor (ISP). Therefore there are 8 ISPs in each 64-way subsystem. There are also 8 channels in each 64-way subsystem. The ISPs and the channels are fully connected, while the processors and the ISPs are not. So any processor can access any disk on any channel within its (64-processor) subsystem through *its* ISP.

Each channel supports an I/O bandwidth of 3 MB/sec. The aggregate data transfer rate for the 512-processor RP3 with 64 channels would be 192 MB/sec.

Appendix C

Split-Step Computation Structure

```

/* RESQ code for modelling split-step migration on RP3
   This listing includes the Computation structure and
   calls the I/O system submodels */
MODEL:COMPUTAT /* Model name */
  PROMPTING LEVEL:2
  METHOD:simulation
  NUMERIC PARAMETERS:pwrntenum /* number of p planes before write*/
  NUMERIC PARAMETERS:writenum /* number of w planes before write*/
  NUMERIC PARAMETERS:jtokens /* number of w planes in stg */
  NUMERIC PARAMETERS:ptokens /* number of processors */
  NUMERIC PARAMETERS:parallel /* number of procs applied to FFT */
  NUMERIC PARAMETERS:procmem /* amt of mem per proc in MB */
  NUMERIC PARAMETERS:procmips /* Processor speed in MIPS */
  NUMERIC PARAMETERS:wmax /* Number of w planes */
  NUMERIC PARAMETERS:pmax /* Number of P or V planes */
  NUMERIC PARAMETERS:iosysnum /* number of 8 isp clusters(1 to 8*/
  NUMERIC IDENTIFIERS:btokens memsize planedim
/*Number of buffer tokens is total system memory / size of velocity
  plane. Request for w or pr will get 2 and 1 tokens respectively */
  BTOKENS: ptokens*memsize/(planedim*planedim*4) /* vel plane */
  MEMSIZE:procmem*1024*1024 /* calculate system mem in bytes */
  PLANEDIM:512
  NUMERIC IDENTIFIERS:initime
  INITIME:.0000001
/* Time values for the three processing steps.... */
  NUMERIC IDENTIFIERS:threetime phitwotime sigmatime
  THREETIME:67.7/procmips /* times for base speed of 1 mip */
  PHITWOTIME:6.2/procmips
  SIGMATIME:0.14/procmips
  NUMERIC IDENTIFIERS:vdelaytime pdelaytime /* Delays to order I/O */
  VDELAYTIME:.0000001 /* requests correctly */
  PDELAYTIME:.0000002
  INCLUDE:const
  GLOBAL VARIABLES:indexnum totplane
  INDEXNUM:1
  TOTPLANE:0
  GLOBAL VARIABLES:puse(pmax) pbusy(pmax)
  PUSE:0 /* Usage of p and v planes */
  PBUSY:0 /* Busy flags for p planes */
  GLOBAL VARIABLES:winmem(wmax) vinmem(pmax) pinmem(pmax)
  WINMEM:0 /* In-memory status for planes */
  VINMEM:0
  PINMEM:0
  MAX JV:15
  QUEUE:initq
  TYPE:is
  CLASS LIST:ini
  SERVICE TIMES:initime
  QUEUE:processorg
  TYPE:passive
  TOKENS:ptokens /* Total number of processors */
  DSPL:fcfs

```

```

    ALLOCATE NODE LIST:ap3q apphi2q apsigmaq
    NUMBERS OF TOKENS TO ALLOCATE:parallel 1 1 /* parallel FFT */
    RELEASE NODE LIST:rp3q rpphi2q rpsigmaq
QUEUE:jobq
    TYPE:passive
    TOKENS:jtokens /* number of jobs in system */
    DSPL:prty
    ALLOCATE NODE LIST:aj
    NUMBERS OF TOKENS TO ALLOCATE:1
    PRIORITIES:jv(iter) /* Higher priority for jobs with lower*/
    RELEASE NODE LIST:rj1 rj2 /* iterations*/
QUEUE:bufferq
    TYPE:passive
    TOKENS:btokens
    DSPL:fcfs
    ALLOCATE NODE LIST:abw abv abp/* w plane needs 2 slots*/
    NUMBERS OF TOKENS TO ALLOCATE:2 1 1/* v & p need 1 each */
    DESTROY NODE LIST:dest
    CREATE NODE LIST:createw createp
    NUMBER OF TOKENS TO CREATE:2
QUEUE:threeq
    TYPE:is
    CLASS LIST:three
    /* Adjust FFT time for parallel processing */
    SERVICE TIMES:(threetime/TH(processorq))
QUEUE:phitwoq
    TYPE:is
    CLASS LIST:phitwo
    SERVICE TIMES:phitwotime
QUEUE:sigmaq
    TYPE:is
    CLASS LIST:sigma
    SERVICE TIMES:sigmatime
QUEUE:vdelayq
    TYPE:is
    CLASS LIST:vdelay
    SERVICE TIMES:vdelaytime
QUEUE:pdelayq
    TYPE:is
    CLASS LIST:pdelay
    SERVICE TIMES:pdelaytime
SET NODES:giveindex
    /* Allot each job its index number */
    ASSIGNMENT LIST:jv(index)=indexnum indexnum=indexnum+1++
    jv(iter)=1
    /* Set parameters for different I/O requests */
SET NODES:wread
    ASSIGNMENT LIST:jv(read)=1 jv(item)=1 jv(len)=planedim*planedim*8
SET NODES:vread
    ASSIGNMENT LIST:jv(read)=1 jv(item)=2 jv(len)=planedim*planedim*4
SET NODES:pread
    ASSIGNMENT LIST:jv(read)=1 jv(item)=3 jv(len)=planedim*planedim*4
SET NODES:wwrite
    ASSIGNMENT LIST:jv(read)=0 jv(item)=1 winmem(jv(index))=0 ++
    jv(len)=planedim*planedim*8
SET NODES:pwrite
    ASSIGNMENT LIST:jv(read)=0 jv(item)=3 vinmem(jv(iter))=0 ++
    jv(len)=planedim*planedim*4 pinmem(jv(iter))=0
/* Increment usage of p planes to check completion of iterations*/
SET NODES:incpuse
    ASSIGNMENT LIST:puse(jv(iter))=puse(jv(iter))+1 ++
    totplane=totplane+1
SET NODES:inciter1 inciter2 /*increment iteration # for next pass*/
    ASSIGNMENT LIST:jv(iter)=jv(iter)+1
/* Provide mutually exclusive access to p planes using busy flags*/
SET NODES:setpbusy

```

```

    ASSIGNMENT LIST:pbusy(jv(iter))=1
SET NODES:resetpbusy
    ASSIGNMENT LIST:pbusy(jv(iter))=0
    /* Indicate data transfer into memory with "1" being in memory */
    /* and "2" for being transfered, "0" is for not in memory */
SET NODES:setwinmem
    ASSIGNMENT LIST:winmem(jv(index))=1
SET NODES:setvinmem
    ASSIGNMENT LIST:vinmem(jv(iter))=1
SET NODES:setpinmem
    ASSIGNMENT LIST:pinmem(jv(iter))=1
SET NODES:vreq
    ASSIGNMENT LIST:vinmem(jv(iter))=2
SET NODES:preq
    ASSIGNMENT LIST:pinmem(jv(iter))=2
WAIT NODES:mux /* Mutually exclusive access to p planes */
    PREDICATE LIST:until(pbusy(jv(iter))=0)
    /* Wait for data transfer to be completed for v and p planes */
WAIT NODES:waitforv
    PREDICATE LIST:until(vinmem(jv(iter))=1)
WAIT NODES:waitforp
    PREDICATE LIST:until(pinmem(jv(iter))=1)
SPLIT NODES:sp11 spl2
FISSION NODES:fis1 fis2 fis3
FUSION NODES:fus1 fus2 fus3
DUMMY NODES:dum1 dum2 ioreq ioack wack pack
**/
/* To test computation model without IO overhead use INCLUDE: io2 */
INCLUDE: io
INVOCATION: ios
    TYPE:io
    C:compchain
CHAIN:compchain
    TYPE:open
    :ini->giveindex->aj->fis1
    :fis1->fus3 fis2 pread;fission
    :pread->pdelay ;if(pinmem(jv(iter))=0)
    :pread->fus3 ;if(pinmem(jv(iter))=1)
    :pread->waitforp ;if(pinmem(jv(iter))=2)
    :pdelay->preq->abp->dest->ioreq
    :fis2->fus2 fis3 vread;fission
    :vread->vdelay ;if(vinmem(jv(iter))=0)
    :vread->fus2 ;if(vinmem(jv(iter))=1)
    :vread->waitforv ;if(vinmem(jv(iter))=2)
    :vdelay->vreq->abv->dest
    :fis3->fus1 wread;fission
    :wread->abw fus1 ;if(winmem(jv(index))=0) if(t)
    :abw->dest
    :waitforv->fus2
    :waitforp->fus3
    :fus1->ap3q->three->rp3q->fus2
    :fus2->apphi2q->phitwo->rpphi2q->fus3
    :fus3->mutex->setpbusy->apsigmaq->sigma->rpsigmaq->resetpbusy
    :resetpbusy->incpuse
    :incpuse->sp11 dum1;if(puse(jv(iter)) mod pwritenum=0) if(t)
    :sp11->dum1 pwrite;split
    :pwrite->ioreq
    :dum1->rj1 dum2;if(jv(iter)=pmax) if(t)
    :rj1->sink
    :dum2->rj2 inciter1 ;if(jv(iter) mod wwritenum=0) if(t)
    :rj2->sp12
    :sp12->inciter2 wwwrite;split
    :inciter2->aj
    :wwrite->ioreq
    :inciter1->fis1
    :ioack->wack ;if(jv(item)=1)

```

```

:ioack->setvinmem;if(jv(item)=2)
:ioack->pack      ;if(jv(item)=3)
:wack->setwinmem createw ;if(jv(read)=1) if(t)
:pack->setpinmem createp ;if(jv(read)=1) if(t)
:setwinmem->fus1
:setvinmem->fus2
:setpinmem->fus3
:createw->sink
:createp->sink
:ioreq->ios->ioack
CONFIDENCE INTERVAL METHOD:none
INITIAL STATE DEFINITION -
CHAIN:compchain
  NODE LIST:ini
    INIT POP:wmax
/* use this line to eliminate start up effects
INITIAL PORTION DISCARDED:      (in percent)*/
RUN LIMITS -
LIMIT - CP SECONDS:100
TRACE: no /*
  INITIALLY ON:yes
  TURN TRACE ON -
  TURN TRACE OFF -
  JOB MOVEMENT:yes
  QUEUES:no
  EVENT HANDLING:no
  EVENT LIST:no
  SNAPSHOTS:no*/
END
END

```

Appendix D

I/O Systems

```
/* This file gives a listing of 2 submodels of the I/O system.
The first submodel chooses between striped and unstriped
cases, while the second submodel shows the modeling of the
page buffers, disks, channel etc. */

SUBMODEL: io                                /* Submodel to combine striping*/
CHAIN PARAMETERS: c                        /* submodel with io subsystem. */
DUMMY NODES: d1 d2 din dout
/* Use striping model below for no striping
INCLUDE: stripe2
/* Use striping model below for striping across all 8 isp/stgdir */
/*INCLUDE: stripe*/
INCLUDE: iosys
INVOCATION:iosystem(8)                    /* At most 8 sets of 8 ISPs used */
TYPE:iosys
IO:c
INVOCATION:striping
TYPE:STRIPE
C:c
IOACK:d2
IOREQ:d1
CHAIN:c
TYPE:external
INPUT: din
OUTPUT: dout
:din->striping->dout
/*IO requests for w planes are partitioned evenly across all
sets of ISPs present. All p and v planes are assumed to reside
on ISP set 1.
:d1->iosystem(2) iosystem(3) iosystem(4) iosystem(5) iosystem(6)++
iosystem(7) iosystem(8) iosystem(1);
++
if(jv(item)=wplane and floor(jv(index-1)/(jtokens/iosysnum))+1=2) ++
if(jv(item)=wplane and floor(jv(index-1)/(jtokens/iosysnum))+1=3) ++
if(jv(item)=wplane and floor(jv(index-1)/(jtokens/iosysnum))+1=4) ++
if(jv(item)=wplane and floor(jv(index-1)/(jtokens/iosysnum))+1=5) ++
if(jv(item)=wplane and floor(jv(index-1)/(jtokens/iosysnum))+1=6) ++
if(jv(item)=wplane and floor(jv(index-1)/(jtokens/iosysnum))+1=7) ++
if(jv(item)=wplane and floor(jv(index-1)/(jtokens/iosysnum))+1=8) ++
if(t)
:iosystem(*)->d2
END OF SUBMODEL IO
```

```
/* This submodel models one 64 processor 8 ISP portion of RP3 IO
subsystem. Requests to this submodel contain information about which
ISP, stgdir, and disk, as well as the length of the request and an
indication of read or write.
```

Currently the model assumes a fixed mapping: ISPk only communicates with the corresponding storage director.


```

*/
SUBMODEL:iosys
CHAIN PARAMETERS: io
DUMMY NODES:ind outd
SUBMODEL:ispm
    CHAIN PARAMETERS: c
    NUMERIC IDENTIFIERS:diskperstr /*number of disks per string*/
    DISKPERSTR:3
    NUMERIC IDENTIFIERS:pbsize /*page buffer size */
    PBSIZE:16*1024 /* 16k bytes per page buffer*/
    NUMERIC IDENTIFIERS:numpb /*number of page buffers */
    NUMPB:16/*16 page buffers per ISP */
    NODE ARRAYS:aqchan2(diskperstr)
/* ISP Page Buffer Passive Queue */
    QUEUE:pagebuffer /*Page buffers are passive q */
    TYPE:passive
    TOKENS:numpb
    DSPL:fcfs
    ALLOCATE NODE LIST:aqpb
    NUMBERS OF TOKENS TO ALLOCATE:jv(len)/pbsize
    RELEASE NODE LIST:relpb
/* ISP Page Buffer Passive Queue */
    QUEUE:channel
    TYPE:passive
    TOKENS:1
    DSPL:fcfs
    ALLOCATE NODE LIST:aqchan1 aqchan2(*)
    NUMBERS OF TOKENS TO ALLOCATE:1
    RELEASE NODE LIST:relchan1 relchan2 relchan3
SUBMODEL:disksys
    NODE PARAMETERS:relchan1 relchan2 aqchan21 aqchan22 aqchan23
    CHAIN PARAMETERS:c
    NUMERIC IDENTIFIERS:xferrate xferdelay
    XFERRATE:3*1024*1024 /*3mb/sec */
    XFERDELAY:.001 /* 1 ms transfer delay */
    NODE ARRAYS:aqsd2(diskperstr) xfermiss(diskperstr)
/* Stgdir Passive Queue */
    QUEUE:stgdir
    TYPE:passive
    TOKENS:1
    DSPL:fcfs
    ALLOCATE NODE LIST:aqsd1 aqsd2(*)
    NUMBERS OF TOKENS TO ALLOCATE:1
    RELEASE NODE LIST:relsd1 relsd2
/* Data Transfer Queue */
    QUEUE:xfer
    TYPE:fcfs
    CLASS LIST:xferhit xfermiss(*)
    SERVICE TIMES:jv(len)/xferrate
    DUMMY NODES: dummyin
/* Disk submodel */
    SUBMODEL:diskm
    NODE PARAMETERS:aqchan2 xfermiss aqsd2
    CHAIN PARAMETERS:c
    NUMERIC IDENTIFIERS:seek latency
    SEEK:.016 /* 16 ms seek */
    LATENCY:.008 /*8 ms latency */
    QUEUE:deviceq
    TYPE:passive
    TOKENS:1
    DSPL:fcfs
    ALLOCATE NODE LIST:aqdisk
    NUMBERS OF TOKENS TO ALLOCATE:1
    RELEASE NODE LIST:reldisk
    QUEUE:svctime
    TYPE:fcfs

```

```

CLASS LIST:svc
SERVICE TIMES:seek+latency
CHAIN:c
TYPE:external
INPUT:aqdisk
OUTPUT:reldisk
:aqdisk->svc->aqchan2->aqsd2->xfermiss
:xfermiss->reldisk
END OF SUBMODEL DISKM
INVOCATION:disks1
TYPE:diskm
AQCHAN2:aqchan21
XFERMISS:xfermiss(1)
AQSD2: aqsd2(1)
C:c
INVOCATION:disks2
TYPE:diskm
AQCHAN2:aqchan22
XFERMISS:xfermiss(2)
AQSD2: aqsd2(2)
C:c
INVOCATION:disks3
TYPE:diskm
AQCHAN2:aqchan23
XFERMISS:xfermiss(3)
AQSD2: aqsd2(3)
C:c
CHAIN:c
TYPE:external
INPUT:dummyin
OUTPUT:reld2
:dummyin->aqsd1 relchan1; if(ta>0) if(t)
/* un comment following line for disk cache */
:aqsd1->reld2 xferhit;if(jv(index)=1) if(t)
/* uncomment following line for no disk cache */
/*:aqsd1->reld2
:reld2->relchan2
:relchan2->disks1 ; if(jv(disk) = 1)
:relchan2->disks2 ; if(jv(disk) = 2)
:relchan2->disks3 ; if(jv(disk) = 3)
:disks1 disks2 disks3 ->reld2
:xferhit->reld2
END OF SUBMODEL DISKSYS
INVOCATION:disksys1
TYPE:disksys
RELCHAN1:relchan1
RELCHAN2:relchan2
AQCHAN21:aqchan2(1)
AQCHAN22:aqchan2(2)
AQCHAN23:aqchan2(3)
C:c
CHAIN:c
TYPE:external
INPUT:aqpb
OUTPUT:relpb
:relchan1->aqchan1
:aqpb->aqchan1->disksys1
:disksys1->relchan3;
:relchan3->relpb
END OF SUBMODEL ISPM
INVOCATION:isps(8)
TYPE:ispm
C:io
CHAIN:io
TYPE:external
INPUT:ind

```

```
      OUTPUT:outd
      :ind->isps(1); if(jv(isp)=1)
      :ind->isps(2); if(jv(isp)=2)
      :ind->isps(3); if(jv(isp)=3)
      :ind->isps(4); if(jv(isp)=4)
      :ind->isps(5); if(jv(isp)=5)
      :ind->isps(6); if(jv(isp)=6)
      :ind->isps(7); if(jv(isp)=7)
      :ind->isps(8); if(jv(isp)=8)
      :isps(*)->outd
END OF SUBMODEL IOSYS
```

Appendix E

Striping Schemes

```

/* This appendix has the listing for the striped and unstriped cases*/

/* Striping submodel. This version divides each of the three planes
   into 8 equal blocks and distributes them on disks across each of 8
   ISPs. The three sets of planes reside on different disks.

   A request to the submodel spawns 8 request to the IO subsystem, and
   then waits until all 8 IO requests have been serviced. */
SUBMODEL:stripe
  NODE PARAMETERS: ioack,ioreq
  CHAIN PARAMETERS: c
  SET NODES:s1
    ASSIGNMENT LIST:jv(disk)=jv(item),jv(len)=jv(len)/8,jv(stgdir++
)=1,jv(isp)=1
  SET NODES:s2
    ASSIGNMENT LIST:jv(disk)=jv(item),jv(len)=jv(len)/8,jv(stgdir++
)=2,jv(isp)=2
  SET NODES:s3
    ASSIGNMENT LIST:jv(disk)=jv(item),jv(len)=jv(len)/8,jv(stgdir++
)=3,jv(isp)=3
  SET NODES:s4
    ASSIGNMENT LIST:jv(disk)=jv(item),jv(len)=jv(len)/8,jv(isp)=4++
    jv(stgdir)=4
  SET NODES:s5
    ASSIGNMENT LIST:jv(disk)=jv(item),jv(len)=jv(len)/8,jv(isp)=5++
    jv(stgdir)=5
  SET NODES:s6
    ASSIGNMENT LIST:jv(disk)=jv(item),jv(len)=jv(len)/8,jv(isp)=6++
    jv(stgdir)=6
  SET NODES:s7
    ASSIGNMENT LIST:jv(disk)=jv(item),jv(len)=jv(len)/8,jv(isp)=7++
    jv(stgdir)=7
  SET NODES:s8
    ASSIGNMENT LIST:jv(disk)=jv(item),jv(len)=jv(len)/8,jv(isp)=8++
    jv(stgdir)=8
  FISSION NODES:spl
  FUSION NODES:jn1 jn2 jn3
  DUMMY NODES:outd out dd
  CHAIN:c
    TYPE:external
    INPUT:spl
    OUTPUT:OUT
    : spl->dd s1 s2 s3 s4 s5 s6 s7 s8;fission
    : dd->jn1 ;if(jv(item)=1)
    : dd->jn2 ;if(jv(item)=2)
    : dd->jn3 ;if(jv(item)=3)
    : s1 s2 s3 s4 s5 s6 s7 s8->outd->ioreq
    : ioack->jn1 ;if(jv(item)=1)
    : ioack->jn2 ;if(jv(item)=2)
    : ioack->jn3 ;if(jv(item)=3)
    : jn1 jn2 jn3->out
END OF SUBMODEL STRIPE

```

```

/* This version of the striping submodel divides a request into eight
   blocks (since ISP page buffers are limited) all on the same disk,
   i.e. it represents the no striping case. */
SUBMODEL:stripe
  NODE PARAMETERS: ioack,ioreq
  CHAIN PARAMETERS: c
  SET NODES:s1
    ASSIGNMENT LIST:jv(disk)=jv(item),jv(len)=jv(len)/8,jv(stgdir++)
)=JV(ITEM),JV(ISP)=JV(ITEM)
  SET NODES:s2
    ASSIGNMENT LIST:jv(disk)=jv(item),jv(len)=jv(len)/8,jv(stgdir++)
)=JV(ITEM),JV(ISP)=JV(ITEM)
  SET NODES:s3
    ASSIGNMENT LIST:jv(disk)=jv(item),jv(len)=jv(len)/8,jv(stgdir++)
)=JV(ITEM),JV(ISP)=JV(ITEM)
  SET NODES:s4
    ASSIGNMENT LIST:JV(DISK)=JV(ITEM),JV(LEN)=JV(LEN)/8      ++
                      JV(STGDIR)=JV(ITEM) JV(ISP)=JV(ITEM)
  SET NODES:s5
    ASSIGNMENT LIST:JV(DISK)=JV(ITEM),JV(LEN)=JV(LEN)/8      ++
                      JV(STGDIR)=JV(ITEM) JV(ISP)=JV(ITEM)
  SET NODES:s6
    ASSIGNMENT LIST:JV(DISK)=JV(ITEM),JV(LEN)=JV(LEN)/8      ++
                      JV(STGDIR)=JV(ITEM) JV(ISP)=JV(ITEM)
  SET NODES:s7
    ASSIGNMENT LIST:JV(DISK)=JV(ITEM),JV(LEN)=JV(LEN)/8      ++
                      JV(STGDIR)=JV(ITEM) JV(ISP)=JV(ITEM)
  SET NODES:s8
    ASSIGNMENT LIST:JV(DISK)=JV(ITEM),JV(LEN)=JV(LEN)/8      ++
                      JV(STGDIR)=JV(ITEM) JV(ISP)=JV(ITEM)
  FISSION NODES:spl
  FUSION NODES:jn1 jn2 jn3
  DUMMY NODES:outd out dd
  CHAIN:c
    TYPE:external
    INPUT:spl
    OUTPUT:OUT
      : spl->dd s1 s2 s3 s4 s5 s6 s7 s8;fission
      : dd->jn1 ;if(jv(item)=1)
      : dd->jn2 ;if(jv(item)=2)
      : dd->jn3 ;if(jv(item)=3)
      : s1 s2 s3 s4 s5 s6 s7 s8->outd->ioreq
      : ioack->jn1 ;if(jv(item)=1)
      : ioack->jn2 ;if(jv(item)=2)
      : ioack->jn3 ;if(jv(item)=3)
      : jn1 jn2 jn3->out
END OF SUBMODEL STRIPE

```

Appendix F

Definition of Constants

```
/* This file contains list of constants used in both split-step
   and phase-shift migration models */

NUMERIC IDENTIFIERS:item index iter cubin read disk len isp lblock ++
track stgdir wplane vplane pplane readreq write
ITEM:1          /* velocity, partial result, or omega */
INDEX:2         /* index of plane out of total space */
ITER:3          /* which iteration plane is on */
CUBIN:4
READ:5          /* 1 means a read request for plane */
DISK:6          /* Disk index where for this request */
LEN:7           /* Number of bytes in this request */
ISP:8           /* Which ISP to route request through */
LBLOCK:9        /* Logical block on disk */
TRACK:10        /* Track on disk */
STGDIR:11       /* Which storage director for io req. */
WPLANE:1        /* omega plane */
VPLANE:2        /* velocity plane */
PPLANE:3        /* partial result plane */
READREQ:1
WRITE:0
```

Appendix G

Sample Reply File

```
/* This is a typical reply file for running the models */
64 /*PWRITENUM*/
1024 /* WWRITENUM*/
64 /*JTOKENS */
512 /*PTOKENS=NUMBER OF PROCESSORS*/
8 /* DEGREE OF PARALLELISM WITHIN 3Q*/
4 /* megabytes of memory per processor*/
1 /* PROCESSOR MIP RATE */
512
1024
1 /* NUMBER OF IO SYSTEMS*/
ut
GV(TOTPLANE)
```

Appendix H

Phase-Shift Computation Structure

```

/* RESQ code for modelling phase-shift migration on RP3
This listing includes the Computation structure and
calls the I/O system submodels */
MODEL:COMPUTAT /* Model name */
  PROMPTING LEVEL:2
  METHOD:simulation
  NUMERIC PARAMETERS:pwrtenum /* number of p planes before write*/
  NUMERIC PARAMETERS:wwritenum /* number of w planes before write*/
  NUMERIC PARAMETERS:jtokens /* number of w planes in stg */
  NUMERIC PARAMETERS:ptokens /* number of processors in system */
  NUMERIC PARAMETERS:parallel /* number of procs applied to FFT */
  NUMERIC PARAMETERS:procmem /* amt of mem per proc in MB */
  NUMERIC PARAMETERS:procmips /* Processor speed in MIPS */
  NUMERIC PARAMETERS:wmax /* size of problem (w planes) */
  NUMERIC PARAMETERS:pmax /* size of problem (v and p planes*/
  NUMERIC PARAMETERS:iosysnum /* number of 8 isp groups to use */
  NUMERIC IDENTIFIERS:btokens memsize planedim
/*Number of buffer tokens is total system memory / size of velocity
plane. Request for w or pr will get 2 and 1 tokens respectively */
  BTOKENS: ptokens*memsize/(planedim*planedim*4) /* vel plane */
  MEMSIZE:procmem*1024*1024 /* calculate system mem in bytes */
  PLANEDIM:512
  NUMERIC IDENTIFIERS:initime
  INITIME:.0000001
/* Time values for the three processing steps.... */
  NUMERIC IDENTIFIERS:threetime phitwotime sigmatime
  THREETIME:31.7/procmips /* times for base speed of 1 mip */
  PHITWOTIME:6.2/procmips
  SIGMATIME:0.14/procmips
  NUMERIC IDENTIFIERS:vdelaytime pdelaytime /* Delays to order I/O */
  VDELAYTIME:.0000001 /* requests correctly */
  PDELAYTIME:.0000002
  INCLUDE:const
  GLOBAL VARIABLES:indexnum totplane
  INDEXNUM:1
  TOTPLANE:0
  GLOBAL VARIABLES:puse(pmax) pbusy(pmax)
  PUSE:0 /* Usage of p and v planes */
  PBUSY:0 /* Busy flags for p planes */
  GLOBAL VARIABLES:winmem(wmax) vinmem(pmax) pinmem(pmax)
  WINMEM:0 /* In-memory status for planes */
  VINMEM:0
  PINMEM:0
  MAX JV:15
  QUEUE:initq
  TYPE:is
  CLASS LIST:ini
  SERVICE TIMES:initime
  QUEUE:processorq
  TYPE:passive
  TOKENS:ptokens /* Total number of processors */
  DSPL:fcfs

```



```

ALLOCATE NODE LIST:ap3q apphi2q apsigmaq
  NUMBERS OF TOKENS TO ALLOCATE:parallel 1 1 /* parallel FFT */
RELEASE NODE LIST:rp3q rpphi2q rpsigmaq
QUEUE:jobq
  TYPE:passive
  TOKENS:jtokens /* number of jobs in system */
  DSPL:prty
  ALLOCATE NODE LIST:aj
    NUMBERS OF TOKENS TO ALLOCATE:1
    PRIORITIES:jv(iter) /* Higher priority for jobs with lower*/
  RELEASE NODE LIST:rj1 rj2 /* iterations*/
QUEUE:bufferq
  TYPE:passive
  TOKENS:btokens
  DSPL:fcfs
  ALLOCATE NODE LIST:abw abv abp/* w plane needs 2 slots*/
    NUMBERS OF TOKENS TO ALLOCATE:2 1 2/* */
  DESTROY NODE LIST:dest
  CREATE NODE LIST:createw createp
  NUMBER OF TOKENS TO CREATE:2
QUEUE:threeq
  TYPE:is
  CLASS LIST:three invfft
    /* Adjust FFT time for parallel processing */
  SERVICE TIMES:threetime/TH(processorq)
QUEUE:phitwoq
  TYPE:is
  CLASS LIST:phitwo
  SERVICE TIMES:phitwotime
QUEUE:sigmaq
  TYPE:is
  CLASS LIST:sigma
  SERVICE TIMES:sigmatime
QUEUE:vdelayq
  TYPE:is
  CLASS LIST:vdelay
  SERVICE TIMES:vdelaytime
QUEUE:pdelayq
  TYPE:is
  CLASS LIST:pdelay
  SERVICE TIMES:pdelaytime
SET NODES:giveindex
  /* Allot each job its index number */
  ASSIGNMENT LIST:jv(index)=indexnum indexnum=indexnum+1++
    jv(iter)=1
  /* Set parameters for different I/O requests */
SET NODES:wread
  ASSIGNMENT LIST:jv(read)=1 jv(item)=1 jv(len)=planedim*planedim*8
SET NODES:vread
  ASSIGNMENT LIST:jv(read)=1 jv(item)=2 jv(len)=planedim*planedim*4
SET NODES:pread
  ASSIGNMENT LIST:jv(read)=1 jv(item)=3 jv(len)=planedim*planedim*8
SET NODES:wwrite
  ASSIGNMENT LIST:jv(read)=0 jv(item)=1 winmem(jv(index))=0++
    jv(len)=planedim*planedim*8
SET NODES:pwrite
  ASSIGNMENT LIST:jv(read)=0 jv(item)=3 vinmem(jv(iter))=0++
    jv(len)=planedim*planedim*8 pinmem(jv(iter))=0
/* Increment usage of p planes to check completion of iterations*/
SET NODES:incpuse
  ASSIGNMENT LIST:puse(jv(iter))=puse(jv(iter))+1 ++
    totplane=totplane+1
SET NODES:inciter1 inciter2 /*increment iteration # for next pass*/
  ASSIGNMENT LIST:jv(iter)=jv(iter)+1
/* Provide mutually exclusive access to p planes using busy flags*/
SET NODES:setpbusy

```

```

    ASSIGNMENT LIST:pbusy(jv(iter))=1
SET NODES:resetpbusy
    ASSIGNMENT LIST:pbusy(jv(iter))=0
    /* Indicate data transfer into memory with "1" being in memory */
    /* and "2" for being transfered, "0" is for not in memory */
SET NODES:setwinmem
    ASSIGNMENT LIST:winmem(jv(index))=1
SET NODES:setvinmem
    ASSIGNMENT LIST:vinmem(jv(iter))=1
SET NODES:setpinmem
    ASSIGNMENT LIST:pinmem(jv(iter))=1
SET NODES:vreq
    ASSIGNMENT LIST:vinmem(jv(iter))=2
SET NODES:preq
    ASSIGNMENT LIST:pinmem(jv(iter))=2
WAIT NODES:muxex /* Mutually exclusive access to p planes*/
    PREDICATE LIST:until(pbusy(jv(iter))=0)
    /* Wait for data transfer to be completed for v and p planes */
WAIT NODES:waitforv
    PREDICATE LIST:until(vinmem(jv(iter))=1)
WAIT NODES:waitforp
    PREDICATE LIST:until(pinmem(jv(iter))=1)
SPLIT NODES:sp11 spl2
FISSION NODES:fis1 fis2 fis3
FUSION NODES:fus1 fus2 fus3
DUMMY NODES:dum0 dum1 dum2 ioreq ioack wack pack
/**/
/* To test computation model without IO overhead use INCLUDE: io2 */
INCLUDE: io
INVOCATION: ios
TYPE:io
C:compchain
CHAIN:compchain
TYPE:open
:ini->giveindex->aj->fis1
:fis1->fus3 fis2 pread;fission
:pread->pdelay ;if(pinmem(jv(iter))=0)
:pread->fus3 ;if(pinmem(jv(iter))=1)
:pread->waitforp ;if(pinmem(jv(iter))=2)
:pdelay->preq->abp->dest->ioreq
:fis2->fus2 fis3 vread;fission
:vread->fus2 ;if(vinmem(jv(iter))=0)
:vread->fus2 ;if(vinmem(jv(iter))=1)
:vread->waitforv ;if(vinmem(jv(iter))=2)
:vdelay->vreq->abv->dest
:fis3->fus1 wread;fission
:wread->abw fus1 ;if(winmem(jv(index))=0) if(t)
:abw->dest
:waitforv->fus2
:waitforp->fus3
:fus1->ap3q fus2;if(puse(jv(iter))=wmax) if(t)
:ap3q->three->rp3q->fus2
:fus2->apphi2q->phitwo->rpphi2q->fus3
:fus3->mutex->setpbusy->apsigmaq->sigma->rpsigmaq->resetpbusy
:resetpbusy->incpuse
:incpuse->invfft dum0; if(puse(jv(iter))=wmax) if(t)
:invfft->dum0
:dum0->sp11 dum1;if(puse(jv(iter)) mod pwritenum=0) if(t)
:sp11->dum1 pwrite;split
:pwrit->ioreq
:dum1->rj1 dum2;if(jv(iter)=pmax) if(t)
:rj1->sink
:dum2->rj2 inciter1 ;if(jv(iter) mod wwritenum=0) if(t)
:rj2->sp12
:sp12->inciter2 wwrite;split
:inciter2->aj

```

```

:wwrite->ioreq
:inciter1->fis1
:ioack->wack      ;if(jv(item)=1)
:ioack->setvinmem;if(jv(item)=2)
:ioack->pack      ;if(jv(item)=3)
:wack->setwinmem createw ;if(jv(read)=1) if(t)
:pack->setpinmem createp ;if(jv(read)=1) if(t)
:setwinmem->fus1
:setvinmem->fus2
:setpinmem->fus3
:createw->sink
:createp->sink
:ioreq->ios->ioack
CONFIDENCE INTERVAL METHOD:none
INITIAL STATE DEFINITION -
CHAIN:compchain
  NODE LIST:ini
    INIT POP:wmax
RUN LIMITS -
LIMIT - CP SECONDS:100
TRACE: no /*
  INITIALLY ON:yes
  TURN TRACE ON -
  TURN TRACE OFF -
  JOB MOVEMENT:yes
  QUEUES:no
  EVENT HANDLING:no
  EVENT LIST:no
  SNAPSHOTS:no*/
END
END

```

BIBLIOGRAPHY

- [1] 'The IBM Research Parallel Processor Prototype(RP3): Introduction and Architecture', GF Pfister et al, Proceedings of the International Conference on Parallel Processing, August 1985.
- [2] 'RP3 Processor-Memory Element', WC Brantley, KP McAuliffe and J Weiss, Proceedings of the International Conference on Parallel Processing, August 1985.
- [3] ' "Hot Spot" Contention and Combining in Multistage Interconnection Networks', GF Pfister and VA Norton, Proceedings of the International Conference on Parallel Processing, August 1985.
- [4] 'Final Report:Design and Evaluation of External Memory Architecture for Multiprocessor Computer Systems', JC Browne et al, The University of Texas at Austin, December 1987.
- [5] 'The Research Queueing Package Version 2: Introduction and Examples', CH Sauer, EA MacNair and JF Kurose, RA 138, IBM TJ Watson Research Center.
- [6] 'The Research Queueing Package Version 2: CMS Reference Manual', CH Sauer, EA MacNair and JF Kurose, RA 139, IBM TJ Watson Research Center.

VITA

Mr. S.Sivaramakrishnan was born in Trivandrum, India on May 26, 1964. After his school education at Vikaasa Matriculation Higher Secondary School(Madurai), he attended the PSG College of Technology(Coimbatore) where he was awarded the Bachelors Degree in Electrical and Electronics Engineering in 1985. He pursued graduate study at the Department of Electrical and Computer Engineering at The University of Texas at Austin and was awarded the Masters Degree in Engineering in 1988. Mr. Sivaramakrishnan's interests are in Computer Architecture, Parallel Processing and Distributed Systems.

Permanent address: 66 New Colony
Tuticorin
India 628 003

This thesis was typeset¹ with L^AT_EX by the author.

¹L^AT_EX document preparation system was developed by Leslie Lamport as a special version of Donald Knuth's T_EX program for computer typesetting. T_EX is a trademark of the American Mathematical Society. The L^AT_EX macro package for The University of Texas at Austin thesis format was written by Khe-Sing The.