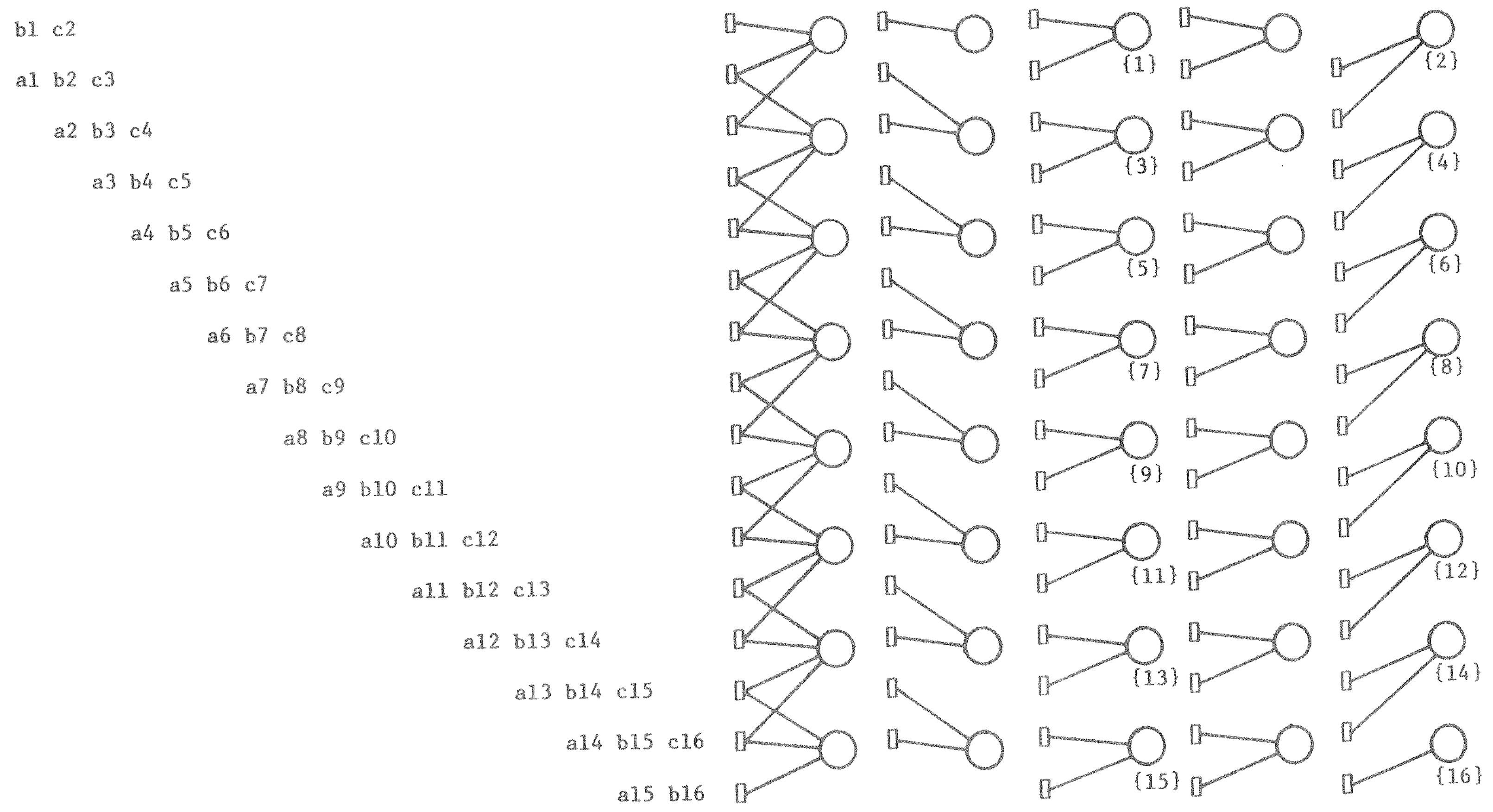


Figure 3.3a: Interconnection for Stage 1, Step 2 of 16x16 Tridiagonal System

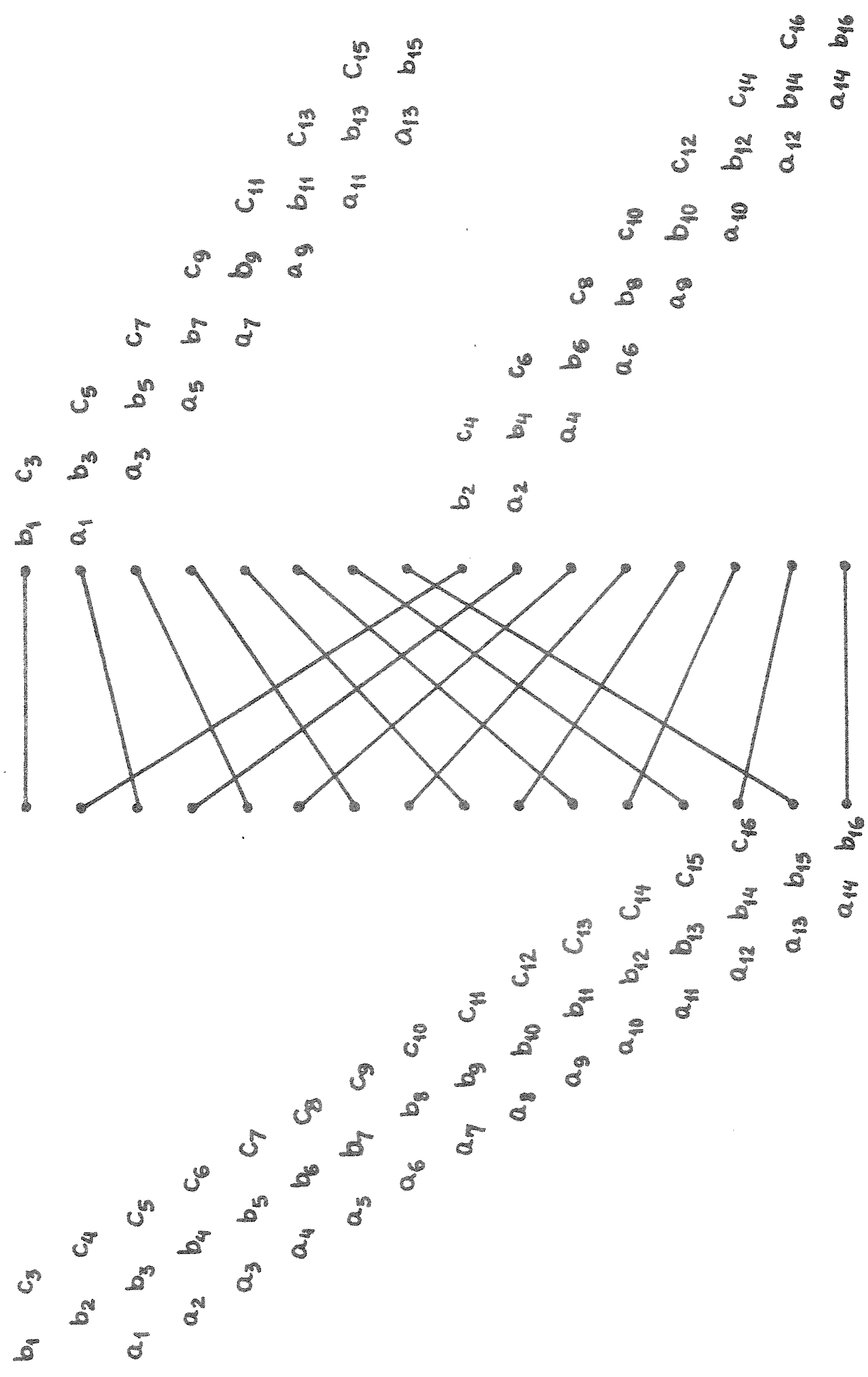


Figure 3.31: Inverse Perfect Shuffle to form Tridiagonal System.

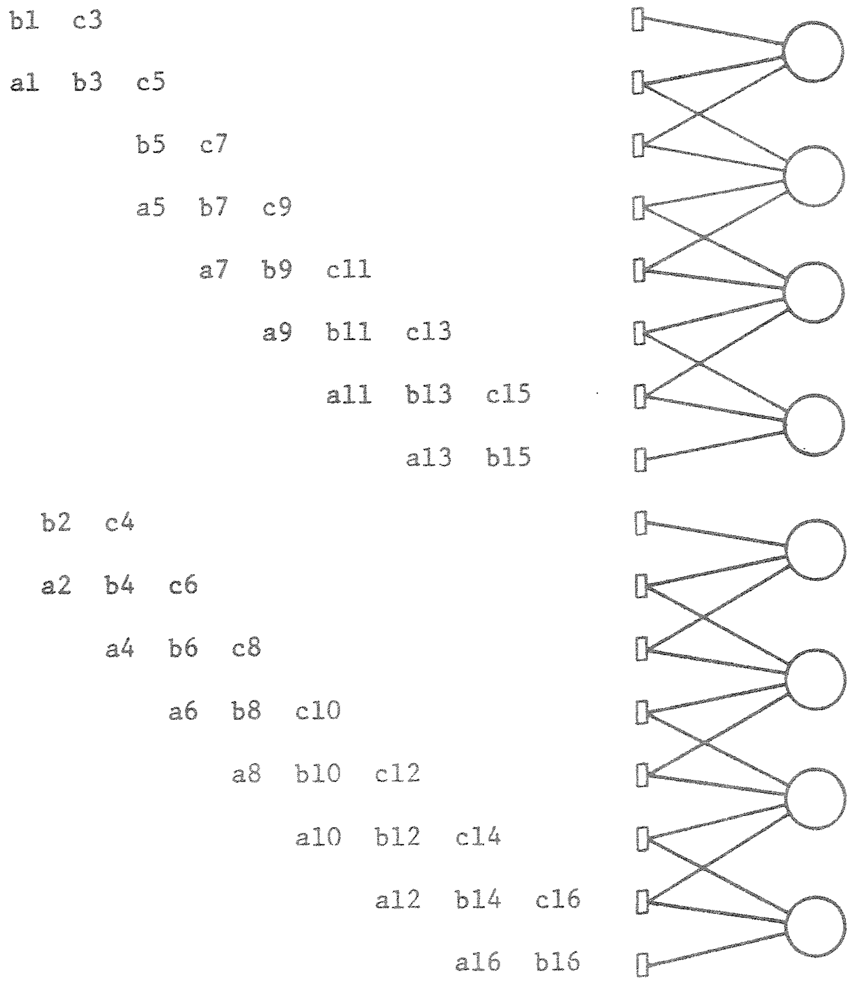


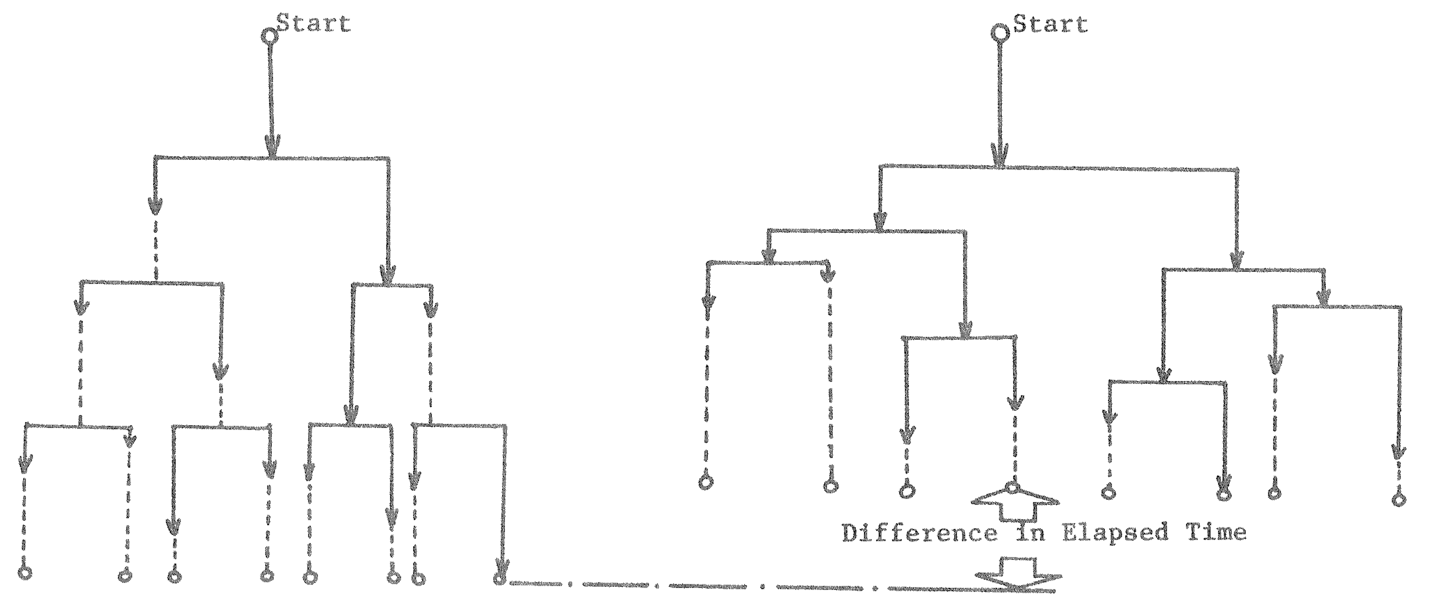
Figure 3.4: Interconnection For Stage 2 Of
Step 2 Of OEE Of 16 X 16 System

3.3.1.2 Intratask Data Flow -

The use of an SISD partition for each block avoids the problem of alternate row/column addressing. Alternate row and column access is necessary because the block matrices, a(k) and c(k) are involved in alternate premultiplication and post multiplication. The use of SIMD partitions would introduce efficiencies in the computational part of the formulation. Data distribution would however, become more complex. Packet movement would be necessary to realign data between pre- and post-multiplication stages. This aspect is not considered any further.

3.3.2 Performance Estimation -

The mode of operation described in the previous subsection consists of asynchronously executing processes which synchronize periodically to transfer data. Operations on different blocks may require different computation times. There may be, for example, different search times for the choice of pivot rows. Additionally, in the case where packets are used, a further dimension of nondeterminacy is introduced. Thus for a performance model to accurately represent this kind of behavior, it must be based on nondeterministic time parameters.



(A) Synchronization Wait at End of every stage

(B) Synchronization Wait at End of Final Stage

Figure 3.5: Synchronization Waits

Two modes of operation based on different synchronization requirements were described in the previous subsection. The mode with explicit synchronization at the end of every stage is easier to control: here all streams of control must wait until the longest stream in the stage is done. The entire elimination, then, is the sum of the lengths of the longest stream in every stage (Figure 3.5a). The second approach allows streams to proceed independently; synchronization is only required at points of data dependency. Here the total execution time is the length of the longest stream chain; this is usually less than the time in the first case (Figure 3.5b).

We will now present a naive analysis of the first mode as a step towards developing a performance model of reconfigurable computer operation. This analysis will first be given for the switched memory implementation and then for the packet implementation.

The time for data movement depends upon the precision of the processors and the precision of the arithmetic. A definite configuration is chosen to illustrate the magnitude of the communication costs. A 16 bit wide SISD partition will be assigned to each block. (If blocks are of uneven size a wider partition could be assigned to larger blocks.) Arithmetic will be on floating point numbers with 64-bit mantissas and 8-bit exponents.

Let the system be 16x16 blocks and each block be 8x8 (total matrix dimension 128x128). Each 8x8 block requires about 600 words of storage. A block row consisting of three 8x8 non zero blocks and a 8x1 vector requires about 2000 words of storage.

The following notation is used for representing operation times:

T.fpa:	floating point addition
T.fpm:	floating point multiply/divide
T.xfer:	memory to memory transfer time for one word
T.swi:	acquisition and setup time to obtain shared memory.
T.pkt:	minimum byte transfer time using packets

The evaluation of H.5 from H.1 proceeds in 4 stages with each stage evaluating H.i+1 from H.i. The first step of a stage consists of the LU factorization of b which is used to evaluate a,c and v. The second step consists of three substeps that correspond to the three positions of the 2-pole, 3-position switch. The final step performs the inverse perfect shuffle to position data for the next stage. From the discussion of the previous subsection it is evident that the first and last step display 16 way parallelism and the second step 8 way parallelism.

H.5 is finally solved as 16 uncoupled linear systems to obtain the required solution.

We will use the notation Pi(k,l,m..) to represent the state where partition Pi is connected to datasets k,l,m.. and dataset k contains a(k),b(k),c(k) and v(k).

The timing for the implementation with shared memory is now presented.

STAGE 1:Compute H.2 from H.1

Step 1:

Configuration: Pa(1), Pb(2),.... , Pp(16)

Setup time ≈ T.swi.

Substep 1.1

Computation: Pa: b(1) (LU decomposition)

Pb: b(2)

.

.

Pp: b(16)

Compute time ≃ 200*T.fpm + 200*T.fpa

Substep 1.2

Computation: Pa: $\underline{a}(1)$, $\underline{c}(1)$, $\underline{v}(1)$

Pb: $\underline{a}(2)$, $\underline{c}(2)$, $\underline{v}(2)$

.

.

Pn: $\underline{a}(16)$, $\underline{c}(16)$, $\underline{v}(16)$

Compute time ≈ 1200*T.fpm + 1200*T.fpa

Step 2

Substep 2.1

Configuration: Pa(1), Pc(2,3), Pe(4,5)

....., Po(14,15)

Setup Time ≃ 2*T.swi

Substep 2.2

Configuration: Pa(1,2), Pc(3,4), Pe(5,6),

.......,Po(15,16)

Setup time ≃ 2*T.swi.

```
Computation:    Pa:     b(1).2, a(1).2,
                        c(1).2, v(1).2
                Pc:     b(3).2, a(3).2,
                        c(3).2, v(3).2
                .
                .
                .
```

Compute time ≃ 2000*T.fpm + 2000*T.fpa

Substep 2.3

Configuration: Pa(1,2), Pc(3,4), Pe(5,6)

......., Po(15,16)

Setup time ≃ 0*T.swi

Substep 2.4

Configuration: Pa(2,3), Pc(4,5), Pe(6,7),

......., Po(16)

Setup Time: ≃ 2*T.swi

Computation:	Pa:	b(2).2, a(2).2,
		c(2).2, v(2).2
	Pc:	b(4).2, a(4).2,
		c(4).2, v(4).2
	.	
	.	
	.	

Compute Time ≃ 2000*T.fpm + 2000*T.fpa

Step 3

Substep 3.1

```
Configuration:  Pa(1),  Pb(2),  Pc(3),  Pd(4),
                Pe(5),  Pf(6),  Pg(7),  Ph(8),
                Pi(9),  Pj(10), Pk(11), Pl(12),
                Pm(13), Pn(14), Po(15), Pp(16)
```

Setup time ≈ T.swi

Computation: Transfer source dataset contents to local buffer.

Compute time ≈ 2000*T.xfer

Substep 3.2

```
Configuration:  Pa(1),  Pb(9),  Pc(2),  Pd(10),
                Pe(3),  Pf(11), Pg(4),  Ph(12),
                Pi(5),  Pj(13), Pk(6),  Pl(14),
                Pm(7),  Pn(15), Po(8),  Pp(16)
```

Setup time ≃ T.swi

Computation: Copy local buffer contents to target dataset.

Compute time ≈ 2000*T.xfer

The implementation using packets avoids a large fraction of the configuration setup time; instead, large volumes of data must be moved within steps (specifically in step 2), between steps and between stages. The following timing calculations are based on the assumption that the average packet transfer time is 2*T.pkt (T.pkt is the minimum packet transfer time - this is seen when no collisions occur. Actual transfer times are related to parameters such as the nature of the background traffic, the actual source, destination traffic pattern in the program and the physical placement of the source memories and the target processors). The computation times remain precisely the same as before.

The following are the setup and data transfer times for the packet implementation:

Step 1:

Setup time ≈ T.swi

Transfer ≈ 4000*T.pkt

Step 2:

Setup time ≈ 2*T.swi

Transfer ≈ 8000*T.pkt

Step 2.1

Transfer ≈ 4000*t.pkt

Step 2.2

Transfer $\simeq$ 4000*T.pkt

Step 2.3

Transfer $\simeq$ 0*T.pkt

Step 2.4

Transfer $\simeq$ 4000*T.pkt

Step 3

Setup time $\simeq$ T.swi

Transfer $\simeq$ 8000*T.pkt

The important concern is the ratio of direct computation time to the sum of the total non computation time (this consists of the various T.swi, T.xfer, T.pkt, synchronization times etc.). Let us make a reasonable assumption that the ratio of execution time for T.xfer|T.pkt|T.fpa|T.fpm|T.swi are 1|3|10|100|1000 and let T.xfer be one microsecond. Estimate the set-up time for T.fpa,T.fpm and T.xfer to be equal to the arithmetic execution time.

For the shared memory implementation the per stage direct computation time is 1188 milliseconds (ms) per stage, reconfiguration time is 9 ms and data transfer time is 8 ms. The total runtime for OEE is then comprised of approximately 5000 ms. of direct computation time and 70 ms. of data transfer time. Thus, if synchronization time is zero, then the overhead associated with mapping the odd/even elimination to a parallel basis is about 70/5000 or less than 2%. Synchronization delays result solely from the differences in processing time for each block. For uniform size blocks, processing time differences between blocks will result from differing effort for pivot selection. This should not exceed 1%. Reconfigurable architectures can assign processing partitions with power proportional to block size. (SISD partitions with a factor of 8 variation in power for 64-bit floating point numbers can be constructed on TRAC.) Thus synchronization delays should not be more than 10% of direct execution time. Using this as an upper bound the total overhead cost in this formulation is approximately 12%.

For the packet implementation the per stage compute time remains 1188 ms. The reconfiguration time is 4 ms and the data transfer time is 100 ms. In this case the data transfer and synchronization delays amount to approximately 20% of the direct computation costs. The speedup over a comparable uniprocessor implementation of OEE is approximately N/2 for the circuit switching as well as the packet switching implementations.

3.4 ODD-EVEN REDUCTION

OER can now be derived from the groundwork established in the previous section. Consider an elimination step in the previous section. Suppose we collect even numbered equations into a linear system (A.2)(x.2)=(w.2), where

$$A.2 = (-a(2j)*b^{-1}(2j-1)*a(2j-1),$$

$$b(2j)- a(2j)*b^{-1}(2j-1)*c(2j-1)- c(2j)*b^{-1}(2j+1)*a(2j+1),$$

$$-c(2j)*b\ (2j+1)*c(2j+1))_{N.2}$$

$$x.2 = x(2j)_{N.2}$$

$$w.2 = (v(2j) - a(2j)*b^{-1}(2j-1)*v(2j-1) - c(2j)*b^{-1}(2j+1)*v(2j+1))_{N.2}$$

$$= v.2(2j)_{N.2}$$

$$N.2 = \mathrm{floor}(N/2)$$

This is a reduction step: it is a reduction step because A.2 is half the size of the original coefficient matrix A. Once this new system is solved, we can compute the remaining components of x.1=x by back substitution.

$$x(2j-1) = b^{-1}(2j-1)*(v(2j-1) - a(2j-1)*x(2j-2) - c(2j-1)*x(2j)),$$

where $j = 1,2,\ldots\ldots\ldots, \mathrm{ceiling}(N/2)$

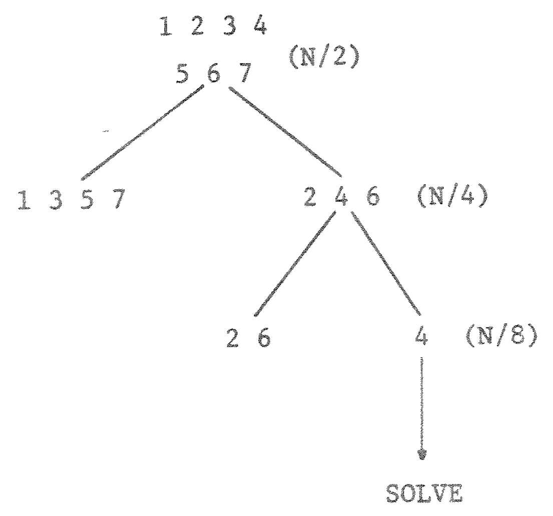


Figure 3.7: Reduction Steps For OER, N = 7

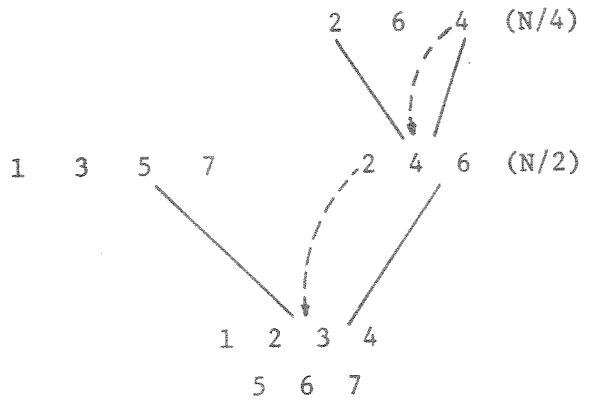


Figure 3.8: Back Substitution For OER

matrix).

In the case of OER (figure 3.7), the number of tasks remains constant from reduction step to step; furthermore, the task size in a given step is half the task size of the previous step. Under the assumption that the reduction step times are equal, the average degree of parallelism is log N.

After the last reduction is performed, exactly one block matrix is left - the solution of this matrix forms the basis for the back substitution sequence. This sequence grows from a parallelism of 2 to N/4 (figure 3.8). This back substitution sequence also shows an average degree of parallelism of log N.

The elimination sequence in OEE (for N = 2**m, figure 3.6) requires one more step than the reduction sequence in OER (for N = 2**m -1, figure 3.7) as indicated by the tree heights in the two figures. OEE does require extraneous operations within each step but these are performed in parallel and so do not contribute to the execution time. After the solution step OER requires an additional back substitution sequence (figure 3.8) that is not needed in OEE. If the back substitution sequence runtime is less than one elimination/reduction stage runtime, OEE is superior to OER. In the example shown latter in this section this is the case for reasonably sized coefficient matrices.

3.4.2 OER Back Substitution Data Flow -

From the back substitution equation of section 3.4, it is evident that each solved block is used in the generation of the solution of two unsolved blocks in each stage of the back substitution sequence. Figure 3.9 shows a configuration suitable for implementing one such stage. Each stage then requires a step with no switched memory access for computing block diagonal inverses; thereafter, switched memory access is necessary in two steps - once for each of the unsolved blocks where the solved blocks are needed.

3.4.3 Performance Estimation For OER -

The computation and data movement costs for OER can be estimated by a procedure similar to that used in subsection 3.3.2 for OEE. We consider only switched memory data movement for OER.

Continuing with the example of subsection 3.3.2, for the reduction sequence the per stage computation time is 1188 ms. and the data transfer time is 17 ms. (this is identical to the elimination sequence per stage times). In the back substitution sequence, the per stage computation time is 240 ms. and the per stage data transfer time is 2 ms. The total computation time for OER is approximately 4500 ms. and the data

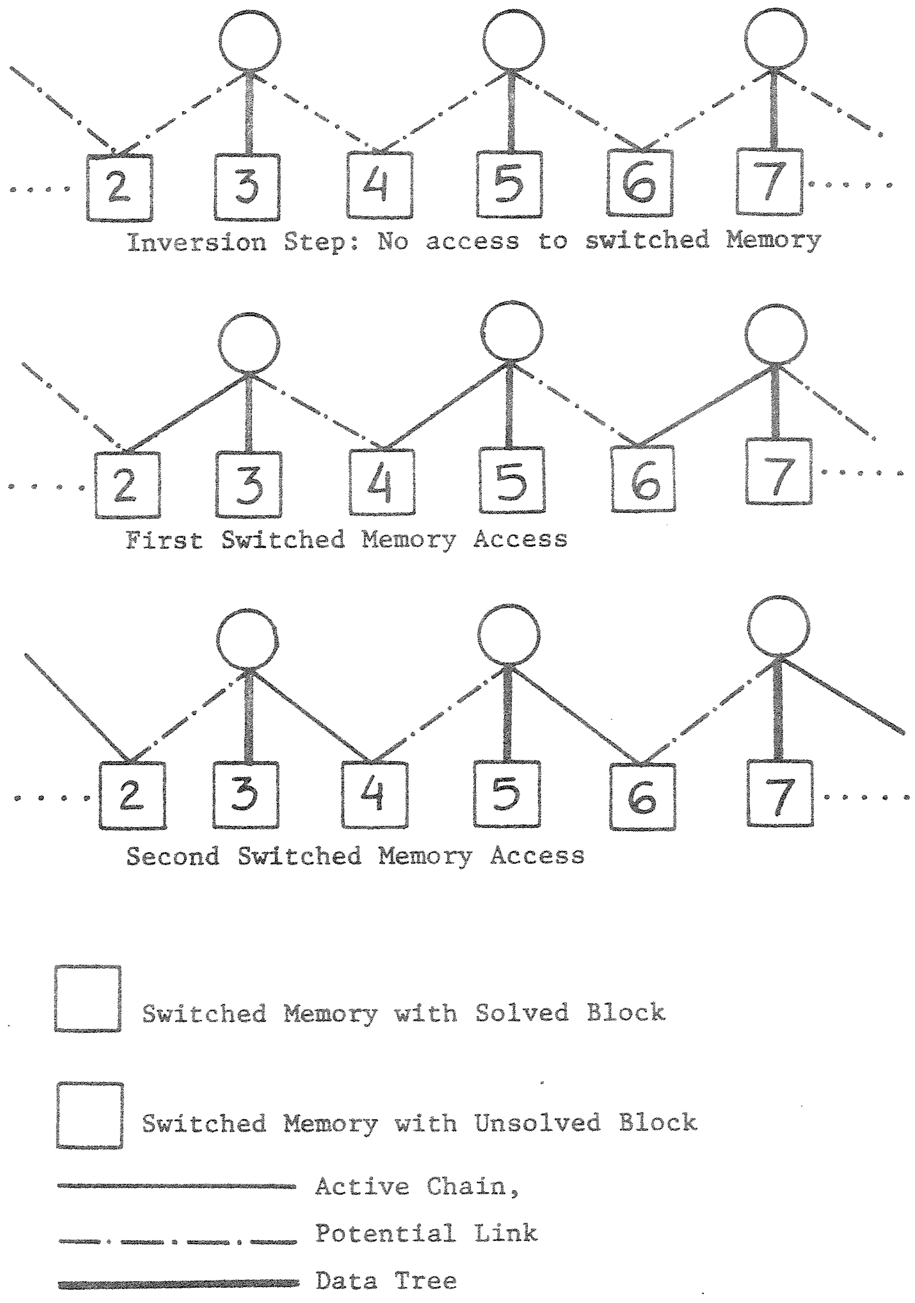

2
3
4
5
6
7
Inversion Step: No access to switched Memory
First Switched Memory Access
Second Switched Memory Access
Switched Memory with Solved Block
Switched Memory with Unsolved Block
Active Chain,
Potential Link
Data Tree

Figure 3.9: Step from Back Substitution
Sequence of OER

transfer time is 60 ms. as opposed to 5000ms. and 70 ms. for OEE. With 64x64 block and larger systems, the back substitution sequence requires more time than an elimination/reduction stage. From this point on, OEE requires less time than OER as shown in figure 3.10. The speedup for OER is approximately log N over a comparable uniprocessor implementation.

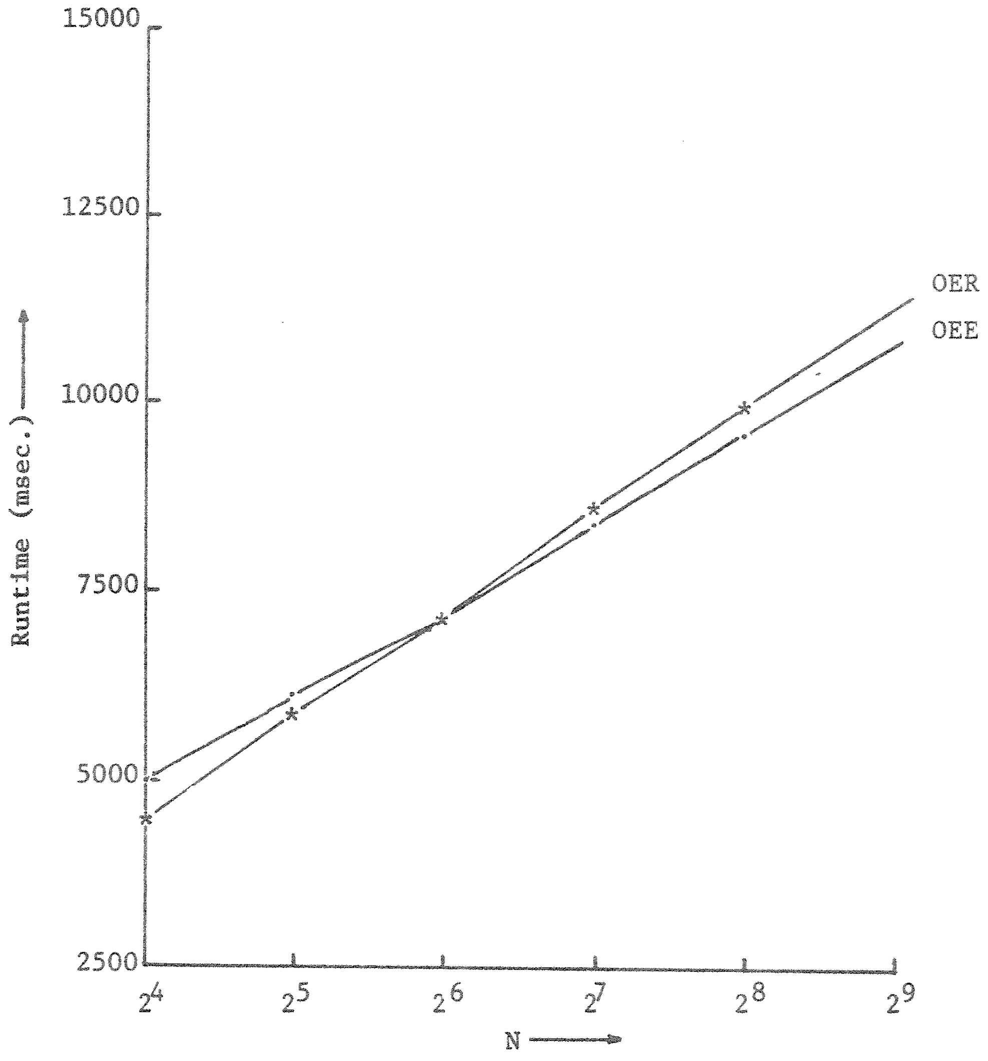


Figure 3.10: OEE & OER Runtime For N X N Block System (8 X 8 Blocks)

3.5 CONCLUSIONS

The technique for programming reconfigurable computers developed in the previous chapter is used to adapt algorithms used on uniprocessor computers to reconfigurable computers. Odd-even elimination and odd-even reduction are cast as multi phase realizations on a representative reconfigurable architecture.

Two issues that have not been considered in this analysis deserve mention. Firstly, this study assumes that configurations can always be set up to completely meet the resource requirements of an executing phase. Aspects of reconfiguring the machine or the algorithm in the presence of insufficient resources have not been considered. Secondly, the issue of the use packet switching versus circuit switching has not been addressed. From a qualitative standpoint, circuit switching fits in naturally with algorithms that repeatedly require block transfers over relatively fixed geometries, as is the case for block OEE and block OER. However, the packet switching implementation is only marginally inferior to the circuit switching implementation of OEE: aspects of overall system utilization may be relevant in a quantitative study of this issue.

Odd-even reduction, which is considered to be a compact form of odd-even elimination on uniprocessor architectures, is projected to have a larger execution time on reconfigurable architectures when the matrix size is moderately large. The speedup over comparable uniprocessor implementations is approximately N/2 fo OEE and log N for OER. This is an instance of the conjecture that many 'good' uniprocessor algorithms are outperformed by 'bad' uniprocessor algorithms on multiprocessor computers.

CHAPTER IV

MODELLING THE EXECUTION OF PARALLEL PROGRAMS ON RECONFIGURABLE ARCHITECTURES

A performance model for the execution of parallel programs on reconfigurable architectures is developed in this chapter. This abstract machine will be used for determining the elapsed execution time of parallel programs and to apportion this time among its component tasks in terms of computation and data movement times and synchronization delays.

4.1 OVERVIEW

The study and development of modelling tools for various aspects of computer systems has received a great deal of attention with the proliferation of the modern multiprogramming operating system. Tools such as discrete event simulation languages [Efr69], analytical queueing modelling systems [Inf81], hybrid simulators [Sch78] etc. are widely available and extensively used in the performance forecasting and tuning of such systems. These tools however, have been shown to be inadequate for modelling concurrent system behavior such as process blocking, holding of multiple resources, etc. [Bro73, Ram80].

The early modelling structures developed for the study of concurrent systems focused on properties such as correctness, termination, deadlock freedom etc. However, one of the major motivations for developing concurrent systems is performance: therefore the need exists for a modelling system that records the passage of time and generates performance metrics such as total execution times, resource holding times and synchronization delays. A few models of computation have been developed for extracting the performance characteristics of concurrent systems. Most of these models have had their formal basis in graph models such as Petri nets.

A review of previous work on structural and time-resolved models of concurrent systems is presented in the following sections as an introduction to the development of a modelling system for the execution of parallel programs on reconfigurable architectures. Section 4.2 contains a brief description of Petri net [Pet77] and Vector Replacement System [Kel74] notation. Section 4.3 contains some examples of timed models for concurrent systems. Section 4.4 provides the formal specifications for a Parallel-program Reconfigurable-architecture Performance (PRP) model. This performance model has been specifically designed for evaluating the performance of parallel numerical codes on reconfigurable computers.

4.2 FORMAL MODELS FOR REPRESENTING CONCURRENCY

The notation and the basic aspects of some formal models (Petri Nets and Vector Replacement Systems) for representing concurrency have been used in this development of performance models for parallel programs executing on reconfigurable architectures. This section establishes notation and fundamental concepts for Petri nets [Pet77] and Vector Replacement Systems [Kel74]. Set Operation Systems (SOS), which form the implementation framework for PRP, are developed from Vector Replacement Systems in subsection 4.2.3. SOS use the same algebraic notation as VRS and are similar to colored Petri nets.

4.2.1 Petri Nets -

A Petri net is an abstract formal model of information flow. The primary use of Petri nets is in the modelling of systems where events can occur concurrently, subject to constraints such as frequency and precedence.

Definition: A Petri net is defined as a bipartite directed graph described by the four-tuple

C=(P,T,I,O)

where

P=[p1,p2,..,pn], a set of places, n>=0,

T=[t1,t2,..,tm], a set of transitions, m>=0,

I is the transition input function, I: T-->2**P,

or, I is a subset of PxT,

O is the transition output function, O: T-->2**P

or, O is a subset of TxP

and the sets P and T are disjoint.

The graph C models the static properties of a system. Figure 4.1 shows an example of a Petri net graph.

In this definition, the connecting arcs are defined by the transition input and output functions. For each transition, the input function yields the set of places connected by arcs directed into the transition while the output function yields the set of places connected by arcs directed away from the transition.

Dynamic properties of the system may be modelled by the introduction of another primitive entity called a 'token'. A marking 'u' of a net is an assignment of tokens to places in that net. Tokens reside in places within the net - the number and distribution of tokens may change during the execution of the net. A Petri net executes by firing transitions. A transition may fire when it is enabled. A transition is enabled if each of its input places contains at least one token. The firing of a transition results in the removal of exactly one token from each input place and the addition of one token to

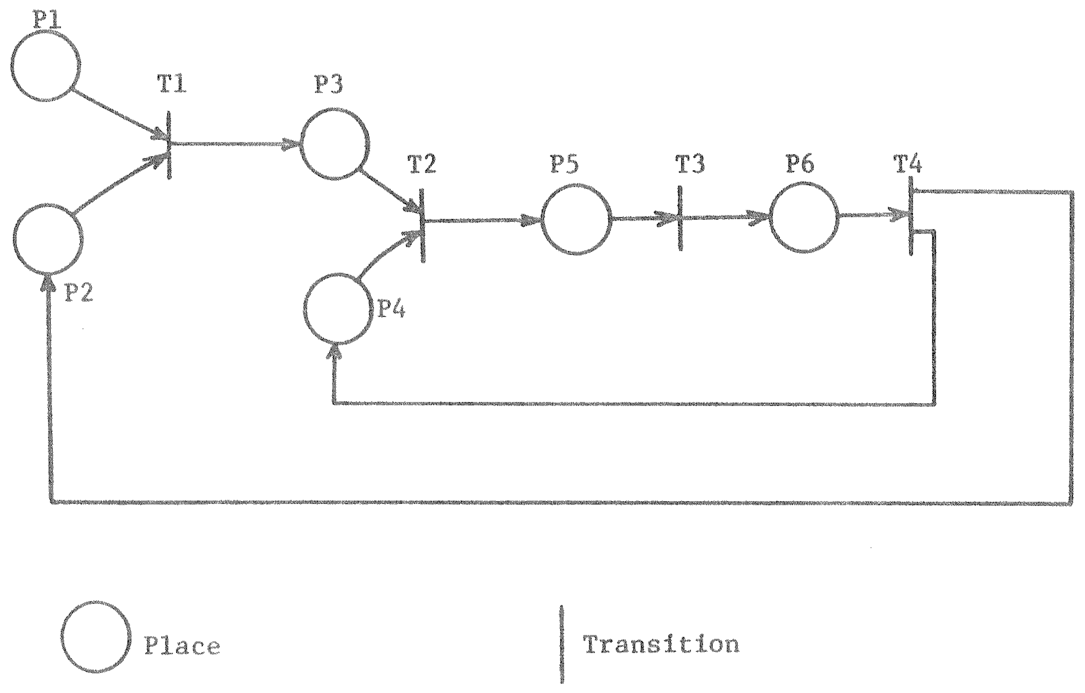


Figure 4.1: Petri Net

each output place (Figure 4.2).

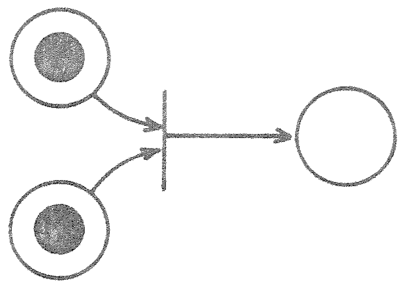
This definition of Petri nets assigns no special meaning to places, tokens, transitions or transition firings; they are primitive objects and actions with no associated attributes or functions. If a Petri net is to be used to model a specific system it is necessary to assign a name and interpretation to each node of the network: this gives rise to 'Interpreted Petri Nets'. Examples of interpreted Petri nets are given in the next section.

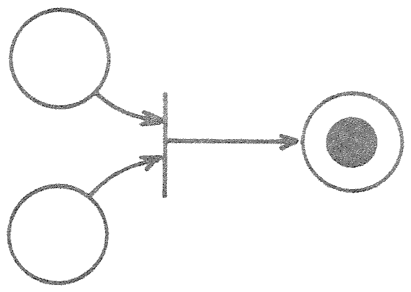
An extension to Petri nets that is potentially useful in compressing net size is the concept of 'colored' or 'typed' tokens [Pet80].

Definition: Let qj and rj be the number of input and output places respectively of a transition tj. A firing rule for transition tj would be a function fj that takes a qj-tuple of input tokens and produces a rj-tuple of output tokens.

Figure 4.3 shows a tabular representation of such a rule. The firing occurs when tokens of appropriate color are present in appropriate input places; it places tokens of specified colors in specified output places.

The formal study of Petri nets has produced a number of results concerning the 'correctness' properties of parallel computation. These results are not of interest in the


Before Firing


After Firing

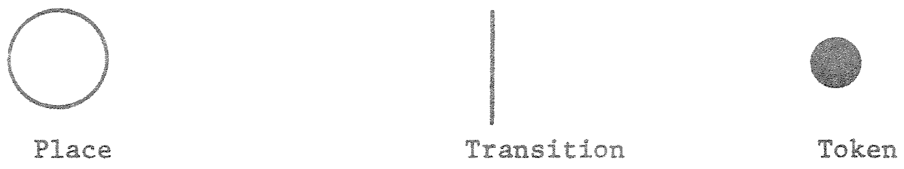
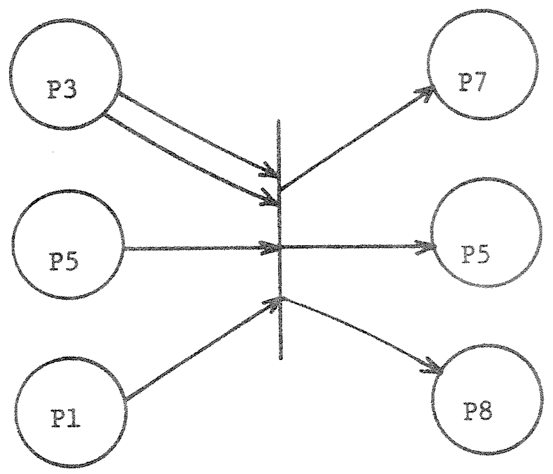


Figure 4.2: Petri Net Firing



INPUTS				OUTPUTS		
P3	P3	P5	P1	P7	P5	P8
Red	Red	Black	Blue	Yellow	Black	Blue
Red	Green	Black	Blue	Orange	Black	Blue
Green	Green	Black	Blue	Red	Black	Blue
.	.	.	.	.	.	.
.	.	.	.	.	.	.

Figure 4.3: Colored Petri Net Firing Rule Definition (from [Pet80])

development of the PRP model because the only goal of this modelling structure is to generate time resolved performance information; a PRP model can be reduced to a Petri net graph if it is necessary to obtain the formal properties of a program.

The semantic structure of Petri nets does embody features needed for PRP but its primitives cannot be cast in a computer representation in the form they are defined. A model that uses an algebraic instead of a symbolic set of primitives is considered next.

4.2.2 Vector Replacement Systems. -

In this section the notation for Vector Replacement Systems is defined; this notation is transformed into Set Operation Systems in subsection 4.2.3.

The notion of a transition system is necessary for introducing the concept of Vector Replacement Systems [Kel74]; Petri nets and Vector Addition Systems [Kar68] are proper subsets of Vector Replacement Systems.

Definition: A **transition system** is a triple (Q,T,->) where
Q is a set of states (not necessarily finite),
T is a finite set of transitions and
-> is a subset of QxTxQ

Let q,q' belong to Q and t to T. Denote (q,t,q') belonging to -> as (q,t)=q'. t transforms state q into q'. It is customary to define a fixed initial state q0; the system is then represented as (Q,T,->,q0).

Definition: Let w=[0,1,2,3......] and Z=[0,1,-1,2,-2,3,-3.....]

Definition: A Vector Replacement System (VRS), (q0,U,V) is a transition system (Q,T,->,q0) in which

1. Q is a subset of w**d for a fixed d called the dimension of the system (d can be thought of as the number of places in the Petri net equivalent of the VRS).

2. T is an arbitrary indexing set so that for each t belonging to T, we have Ut, Vt belonging to Z**d. Ut is called the test vector and Vt is called the replacement vector. Furthermore it is required that Vt>Ut. (If Utp is the p-th coordinate of Ut then Vt>Ut implies that for all p, Vtp>Utp).

3. For q,q' belonging to Q we define (q,t)=q' iff q+Ut>0 and q+Vt=q', where addition is component-wise vector addition.

Note that Ut>Vt and q+Ut>0 guarantees that q' belongs to w**d. Note also that only the non positive components of Ut are of significance.

The Vector Addition Systems (VAS) defined by Karp and Miller [KaMi69] are VRS with Ut=Vt for all t belonging to T [Kel74].

Petri nets are also VRS in which the components of U and V are restricted to 0, 1 and -1 [Kel74]. This correspondence is obvious once the coordinates of vectors are associated with places, vectors with markings and the pair (Ut,Vt) with transitions. The definition of multiple input and output arcs in Petri nets removes the restriction on the components of U and V to 0, 1 and -1.

When VRS is used in the modelling of a specific system, the interpretation of VRS primitives is similar to the interpretation of corresponding Petri net primitives.

We can now define Set Operation Systems (SOS) that use the algebraic primitives of VRS and are similar to colored Petri nets.

4.2.3 Set Operation Systems -

The system that will be now defined is semantically similar to colored Petri nets that allow the movement of distinguishable tokens to and from places when a transition fires. The syntax is based on VRS syntax of the previous subsection and uses an algebraic notation. This forms the

implementation framework for the PRP of section 4.4.

Definition: M=[m1,m2,m3,....mn],a finite set.

N is a subset of M.

Definition: A Set Operation System (SOS), (q0,U,V,R) is a transition system (Q,T,->,q0) in which

1. Q is a subset of N**d for a fixed d called the dimension of the system. (d can be thought of as the number of places in the colored Petri net equivalent of SOS).

2. T is an arbitrary indexing set such that for each t belonging to T we have Ut, Vt and Rt belonging to N**d.

3. For q,q' belonging to Q, we define (q,t)=q' iff
   q intersection Ut = Ut,
   and
   q'= (q minus Vt) union Rt.
   Note:
   Ut is the test 'vector of sets'
   Vt is the extract 'vector of sets'
   Rt is the inject 'vector of sets'.

The operations defined are vector component wise set operations. Figure 4.4 is an SOS that is equivalent to the colored Petri net of Figure 4.3.

$\longleftarrow d \longrightarrow$

	$1^{(3)*}$	$2^{(5)}$	$3^{(1)}$	$4^{(7)}$	$5^{(8)}$
U_1	RED RED	BLACK	BLUE	ϕ	ϕ
V_1	RED RED	ϕ	BLUE	ϕ	ϕ
R_1	ϕ	ϕ	ϕ	YELLOW	BLUE

U_2	RED GREEN	BLACK	BLUE	ϕ	ϕ
V_2	RED GREEN	ϕ	BLUE	ϕ	ϕ
R_2	ϕ	ϕ	ϕ	ORANGE	BLUE

U_3	GREEN GREEN	BLACK	BLUE	ϕ	ϕ
V_3	GREEN GREEN	ϕ	BLUE	ϕ	ϕ
R_3	ϕ	ϕ	ϕ	RED	BLUE

$T = \{1,2,3\}$

* - The number in the paranthesis is the place number in Figure 4.3

Figure 4.4: SOS Equivalent Of Colored Petri Net Of Figure 4.3

4.3 TIMED MODELLING OF CONCURRENT SYSTEMS: A REVIEW

The work described in this section is a further extension (inclusion of time) of the formal models of concurrent behavior described in the previous section. Time extensions are used to obtain metrics such as firing frequencies, subsystem utilization, waiting times, queue lenths etc. The nature of time extensions, the interpretation of the additional primitives and the nature of restrictions placed on the underlying formal model give rise to either a simulation or an analytical modelling system.

The first published simulation modelling system for concurrent systems is due to Noe and his co-workers. He used a Petri net extension for modelling a CDC 6400 operating system [Noe71]. The initial goal of the study was to identify model limitations. This work was continued by Noe and Nutt and the next modelling system was called E-nets [Nut72]; further extensions resulted in a modelling system called Macro-E-nets [Noe73].

E-nets use the same primitives as Petri nets: places (locations), transitions and tokens. Locations are occupied by tokens and represent conditions that may exist for a period of time. Five kinds of transitions are defined- they represent nodes at which determination of token flow and token modification takes place. Macro-E-nets are E-nets with composition facilities for representational ease and clarity.

Berlin proposed and implemented Time Extended Petri Nets (TEPN) [Ber79] for modelling a multi disk multiple controller multiprogrammed computer system. This work also grew out of Petri nets and is similar to E-nets and Macro-E-nets to the extent that the user develops a representation of the system that is being modelled in an appropriate input language; data structures generated from the user representation are exercised to obtain relevant metrics. TEPN uses the same primitives as Petri nets with the following interpretations: tokens are typed and represent conditions, resource requests and resources. Places combine the concept of queues and servers. Functionally, the place receives tokens from input transitions, 'stores' tokens for some elapsed interval and when an appropriate output transition fires, it emits the proper tokens. TEPN transitions serve as synchronization and token modification mechanisms, to control the flow of tokens through the net. Together, these primitives model process service, process blocking and other attributes of parallel systems. Performance metrics are derived from TEPN state attributes.

A number of temporal models that are solved analytically have also been developed. System representations are generated which are amenable to analytic solution to yield metrics such as firing frequencies, queue lengths and others.

Ramamoorthy and Ho [Ram80] ascribe fixed time periods to transition firings; they have developed a technique for determining the minimum cycle time (for processing a task) for decision-free [Pet77] and safe persistent Petri nets [Pet77]. Han [Han78] interprets transitions as service stations, places as queues and tokens as tasks; determination of metrics such as firing frequencies, maximum queue lengths, average queue lengths etc. is done by converting the timed nets into state diagrams and then using Markov chain methods for solution. Ramchandani [Ram73] developed a method for determining maximum firing frequencies for a subclass of timed Petri nets with fixed delays associated with transition firings. Sifakis [Sif77] generalized this work by converting timed Petri nets into a linear system with n unknowns (where n is the number of places) and using that to determine maximum firing frequencies.

Robinson [Rob79] has used a directed graph representation of jobs consisting of asynchronous tasks and developed techniques to determine probability distributions for execution times and bounds on execution times.

4.4 PARALLEL-PROGRAM RECONFIGURABLE-ARCHITECTURE PERFORMANCE

Parallel-program Reconfigurable-architecture Performance (PRP) modelling requirements are as follows:

1. Represent the time resolved behavior of a set of deterministic interacting concurrent tasks.

2. Represent the waiting for and the holding of multiple resources.

3. Allow the specification of the model as a data structure rather than as a custom program.

4. Allow the specification of the user job (user net) in terms of tasks (subnets), as defined in chapter II, and of tasks in terms of simpler constructs.

The fourth requirement arises from the need to tailor the model to handle features that arise from the use of CSL in programming numerical algorithms for reconfigurable computers:

1. Computations display phases: a phase corresponds to a particular configuration and its beginning and end are explicit points of synchronization.

2. There is a large measure of repetitiveness in the geometry of flow of data. This repetitiveness manifests itself as

iterations and recursions along the time and space axes.

PRP is similar in many ways to TEPN and E-nets. Its primitive interpretations, however, are tailored specifically to the modelling of reconfigurable computer mechanisms. Switched memories, tasks and the execution of tasks with captive switched memories are represented in a direct manner.

Definition: A PRP is a modelling structure defined by the four-tuple C = (P,T,I,O), where

P = [p1,p2,.......,pn], a set of places, n>=0,
T = [t1,t2,.......,tm], a set of transitions, m>=0,
I is a transition input function, I:T-->2**P,
or, I is a subset of PxT,
O is a transition output function, O:T-->2**P
or, O is a subset of TxP.
and the sets P and T are disjoint.

The next subsection describes time resolution mechanisms provided in PRP and the following subsection describes management aids provided in PRP for constructing nets.

4.4.1 Parametric Extensions -

This subsection describes and defines the extensions to colored Petri net models that arise from the first three requirements cited for the PRP model; the next subsection describes extensions necessary for fulfilling the fourth requirement. The nomenclature from colored Petri nets and SOS is used interchangeably.

The PRP model is constructed from an SOS by the definition and addition of attributes to the abstract entities used in the SOS definition to allow for time resolved behavior. TEPN [Ber79] primitive interpretations (these have been briefly reviewed in the previous section) are used as a starting point to assign to PRP place and transition, attributes that allow the direct definition of performance metrics as an inherent characteristic of the model. The definition and placement of attributes is predicated upon a performance evaluation viewpoint rather than a theoretical view of Petri nets. The nature of the parametric extensions is such that stripping away these extensions from a PRP leaves an SOS that faithfully reproduces the semantic behavior represented in the PRP net (Appendix B).

The primitive interpretations that follow are based upon and are nominally similar to TEPN [Ber79] primitive interpretations (PRP subattribute definitions within primitive attribute definitions are different from TEPN). Places now represent queues and processes (or wait stations) while transitions remain as synchronization primitives that act as state transition arcs. Transitions can never hold tokens, therefore they are never a part of the state definition of the PRP net. Finally, the interpretation of tokens must allow for typing as well as time stamping. This extended model can represent deterministic token routing; it also provides a mechanism to model the effects of process blocking.

The attributes of the PRP primitives will now be defined; the formal definitions are supported by justifications and motivation where necessary.

Definition: The **PRP place** is a composite entity defined by the following attributes (Figure 4.5):

1. Place wait set: This holds a set of tokens waiting to be 'enabled'.

2. Place enable slot: This holds a single token that is 'enabled', i.e. receiving service in conjunction with tokens from other places (as defined by the input marking of a target transition in the PRP transition definition).

In the context of reconfigurable machines, the place wait set may be used to represent switched memories awaiting attachment; the token in a place enable slot represents an acquired switched memory that is being serviced by a task.

PRP Place:

Place Wait Set: Holds set of unenabled tokens.

Place Enable Slot: Holds enabled token.

Figure 4.5

Definition: The **PRP transition** is a composite entity defined according to the following attributes (Figure 4.6):

1. Input-output mapping set (IOMS):
   IOMS = [IOM / IOM is an input-output mapping]
   IOM = [Input marking, Output marking]
   Input marking = [token type, input place] for all the input places of the transition.
   Output marking = [token type, output place] for all the output places of the transition.

2. Transition delay function set (TDFS): A transition delay function is defined corresponding to each member of IOMS. Each transition delay function is defined as
   TDF = [TDF type, TDF parameters]
   where TDF type can be deterministics, exponential etc. and TDF parameters are parameters appropriate to the TDF type. This function is used to determine the duration a transition remains in the 'enabled' state.