ANALYSIS OF CRAY-1S ARCHITECTURE

VASON P. SRINI
Computer Science Department
University of Alabama
Birmingham, AL 35294

JORGE F. ASENJO
City Bank
Puerto Rico

ABSTRACT

An analysis of the Cray-1S architecture based on dataflow graphs is presented. The approach consists of representing the components of a Cray-1S system as the nodes of a dataflow graph and the interconnections between the components as the arcs of the dataflow graph. The elapsed time and the resources used in a component are represented by the attributes of the node corresponding to the component. The resulting dataflow graph model is simulated to obtain timing statistics using as input a control stream that represents the instruction and data stream of the real computer system.

The Cray-1S architecture is analyzed by conducting several experiments with the model. It is observed that the architecture is a well balanced one and performance improvements are hard to achieve without major changes. Significant improvement in performance is shown when parallel instruction issue is allowed with multiple CIP/LIPs in the architecture.

1. INTRODUCTION

Computer systems such as the CRAY-1S [1,2], CYBER 205 [3], and the Flow Model Processor (FMP) [4] are highly concurrent systems using several advanced architecture principles. In the design of these and other future computer systems, it is important to understand how the system parameters and the work load variations interact. Developing an easily modifiable prototype of a complex computer system and running benchmark programs on it to understand the interaction of parameters is expensive. Analytical models using queuing theory techniques or Markov models, such as the ones presented in [5,6], cannot accurately model state of the art techniques such as conditional issue of instructions, references to interleaved memory, and chaining of vector instructions [1,3]. Monte Carlo simulation models [7] make simplifying and sometimes inaccurate probabilistic assumptions and cannot be used effectively to model complex computer systems.

Dataflow graphs [8,9] are presented as an approach for designing and modeling computer systems with a high degree of concurrency and pipelining. There are two main objectives in using such an approach. The first objective is to analyze the architecture and to detect potential bottlenecks. The second objective is to improve the performance of the system by introducing several changes to the model based on the clues obtained from the analysis. The dataflow approach is motivated by the observation that several of the functional units in CRAY-1S, CYBER 205, and earlier computers such as IBM 360/91 [10,11] and CDC 6600 [10,12] perform in a data driven manner although the whole system is not a dataflow computer system.

The CRAY-1S computer employs many state of the art techniques such as the following:

a. instruction look-ahead,
b. concurrent execution of instructions,
c. pipelining in functional units,
d. interleaved memory,
e. vectorizing and chaining.

The features in each of these areas are accurately modeled using dataflow graphs. Making changes to the model and running experiments on the model can be readily carried out. In our analysis of CRAY-1S, various experiments are performed aimed at improving system performance. These experiments are in the following areas:

a. change the size and number of instruction buffers,
b. eliminate single access path restriction to scalar registers,
c. increase memory bandwidth for vector memory transfers,

d. reduce cycle time of memory banks,
e. extend the period during which chaining of vector operations is allowed to occur, and
f. provide two current instruction parcel registers (CIPs).

To determine the accuracy of the model, and to obtain timing and resource usage statistics necessary to analyze the architecture, the model is simulated using a discrete event simulator such as GPSS [13,14]. Another approach to dataflow simulation is described in [15].

The rest of the paper is divided into three sections. An overview of dataflow graphs is in Section 2. A dataflow graph model of Cray-1S is described in Section 3. Experiments that have been done on the model to analyze the architecture are in Section 4. The complete model of Cray-1S and the control streams used in the experiments are in [16].

2 DATAFLOW GRAPH

The dataflow graph used in this work is an undirected or partially directed graph specified as a 4-tuple (A,N,S,K), where

A is a set of arcs with attributes representing the data paths or the interconnection between the components of a computer system,

N is a set of nodes with attributes representing the components of a computer system including the local controllers and the global controller (score board),

S is a set of special arcs called snoop arcs for gathering the state information from nodes and communicating values to the attributes of arcs and nodes, and

K is a scheme for tagging the tokens on arcs so that several distinct calculations can be processed in the proper order without interference between distinct calculations.

A configuration is an assignment of values to the attributes of arcs and nodes. Initially the graph is in a configuration, I. Changing the configuration of the graph from time to time is called reconfiguration. Although the model has the necessary mechanisms to do dynamic reconfiguration, it will not be emphasized in this work. The interested reader is referred to [9].

As a result of node firing, tokens flow between the nodes in a dataflow graph. Tokens represent the instruction, data, and state information in the real computer system. The number of tokens on the arcs change and the number of nodes firing also change. So, a dataflow graph is a dynamic entity and can model the dynamic aspects of computer systems.

For each arc, several attributes are defined so that some of the parameters of computer systems can be readily represented. The arc attributes are:

< arc label,
node label of one end,
node label of the other end,
arc type ("directed" or "undirected"),
type of token the arc is allowed to carry,
current number of tokens on the arc,
arc capacity,
and arc latency time (the minimum time required for any token to traverse the arc) >.

The first attribute is intended for identification and the next three attributes are for specifying the interconnection characteristics. The fifth attribute is intended for representing the width of data paths or the width of buffers. The sixth attribute is intended to show how many data items are waiting to be processed by the components of a computer system. The arc capacity attribute is intended for specifying the maximum number of data items that can be waiting for a component or the maximum number of buffers. The last attribute is intended for specifying the time required to transfer buffer contents or the memory cycle time. This can be zero or any positive value.

For each node the following attributes are defined:

< node label,
enabling condition (firing semantics set (FSS)),
operation, and
node latency time >

The node label attribute is intended for identification. Sometimes the node label might reflect the operation associated with the node. The FSS is intended for specifying the data items and resources such as buffer space that must be available before the beginning of an activity. The operation attribute is intended for representing the resources used and their state change in performing the transformations on the supplied data. An operation can be a primitive [8] or specified as a dataflow graph. This facilitates a high level or a detailed low level representation for the components in a computer system. The node latency time is provided for specifying the time required to do the transformations on the supplied data. The time can be zero or any positive value.

Using the arc latency time and node latency time the timing statistics for different programs can be calculated provided the instructions of the programs and the data flow generated by the instructions can be supplied as tokens to the dataflow graph. This is done using the control stream approach discussed in [17].

3 MODEL OF CRAY-1S

A Cray-1S computer [1,2] consists of 13 pipelined functional units divided into four groups: scalar, floating point, vector, and address. Several functional units can be concurrently executing instructions depending on the application program and the way in which the instructions are issued.

In this section, a dataflow graph model of Cray-1S is shown. The model is presented at various levels of detail. The high level model comprises five dataflow graphs that represent the following:

a. Fetching and the issuing of instructions,
b. Functional units performing address calculations,
c. Functional units performing operations on scalars,
d. Functional units performing operations on floating point numbers,
e. Functional units performing operations on vectors.

The dataflow graphs are shown in Figures 1 through 5. Some of the key parts of Cray-1S are the instruction fetch and the instruction issue mechanisms. The details of the mechanisms are shown by refining the nodes in Figure 1.

To analyze the architecture of Cray-1S for improving performance and to determine the accuracy of the model, timing statistics and resource usage statistics are needed. The required statistics can be readily obtained if the model is simulated using a deterministic discrete event simulator such as GPSS [13,14] or a dataflow simulator [15]. We decided on GPSS because of its widespread use in simulation and its availability. Furthermore, the primitive nodes in dataflow graphs can be readily represented using the GPSS V function blocks. The attributes of nodes and arcs can be specified as transaction parameters. The nodes in the dataflow graph are realized using GPSS V blocks or PL/1 HELP blocks. The control stream representing the flow of instructions and data in the real computer system are presented as transactions. The details of the dataflow graph model of Cray-1S are now shown.

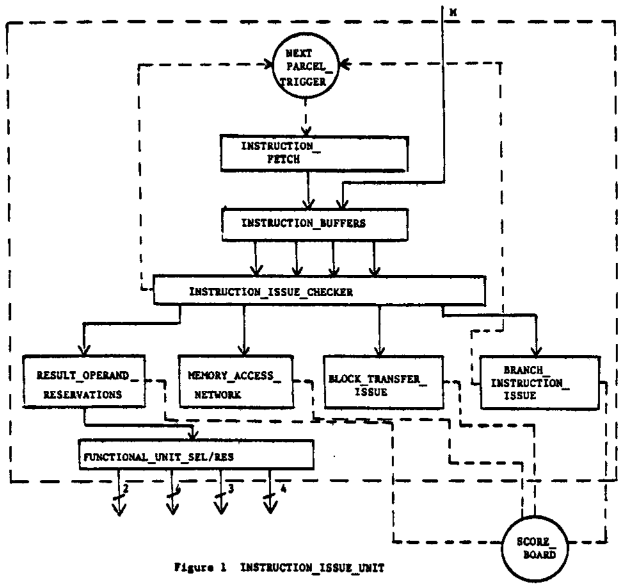

Figure 1 INSTRUCTION_ISSUE_UNIT

INSTRUCTION ISSUE UNIT

This unit identifies the next instruction to be processed, checks the issue conditions against the available resources, and communicates with the global control unit (SCORE_BOARD) to reserve functional units, registers, and data paths. Figure 1 is a dataflow graph of the unit. Each node in Figure 1 is now explained.

The INSTRUCTION_FETCH represents the following:

a. Program counter (P register),
b. Mechanisms to determine whether the next instruction parcel is in the current buffer, another buffer, or in memory (out of buffer), and
c. Mechanisms to initiate instruction fetch from memory on encountering an out of buffer condition.

Figure 1.a shows a dataflow graph of the INSTRUCTION_FETCH. The mnemonics and the notations used in the dataflow graphs are explained in Table 1. Whenever the node P receives a token, it outputs the address of an instruction parcel that may or may not be in the current buffer. The FETCH_MONITOR node determines whether the address received on its second input arc corresponds to the address of an instruction in the current buffer, another buffer, or out of buffer using the token on the first input arc. If the received address is in the range of addresses of the current buffer then the FETCH_MONITOR outputs a token containing the address on the first output arc that eventually reaches the input arc of INSTRUCTION_BUFFER without any time delay. If the received address is in the range of addresses of another buffer, then the FETCH_MONITOR outputs a token on the second output arc. The four buffers are searched in a circular fashion and a time delay of 2 clock periods (CPs) is introduced by the SWITCH_DELAY node. The nodes BSW_SET and BSW_RESET are provided to communicate the buffer switch (BSW) state information with the SCORE_BOARD. This is done so that for two parcel instructions with the lower parcel in another buffer, the upper parcel is held in the NIP until the lower parcel is brought into LIP before issuing the instruction. The third output arc of FETCH_MONITOR receives a token if the address of the instruction is not in the range of addresses of the four buffers. The path emanating from the third output arc represents the fetch sequence from memory. The nodes FRQ_SET, FRQ_RESET, FOP_SET, and FOP_RESET are included to communicate the fetch request (FRQ) and the fetch operation (FOP) state information with the SCORE_BOARD. An instruction fetch sequence from memory cannot proceed in Cray-1S if one of the following is in progress: a scalar memory transfer, B-register or T-register transfer, or a vector block transfer. The above three situations are represented by the nodes SCALAR_DELAY, BTT_DELAY, and VBT_DELAY respectively. The time delay involved in initiating the instruction fetch sequence depends on the type of transfer in program and this is known to the SCORE_BOARD. Hence, snoop arcs are drawn from the SCORE_BOARD to the above three nodes to communicate the time delay. Once an instruction fetch sequence is initiated, there is a time delay of 11 CPs before the instructions are moved into the buffer pointed by the next buffer pointer. This is represented by the FETCH_SEQUENCE_DELAY node.

The four buffers of the Cray-1S are represented by the INSTRUCTION_BUFFERS node. On receiving a token on the first input arc containing the address of an instruction, the contents of the address are supplied on one of the four output arcs.

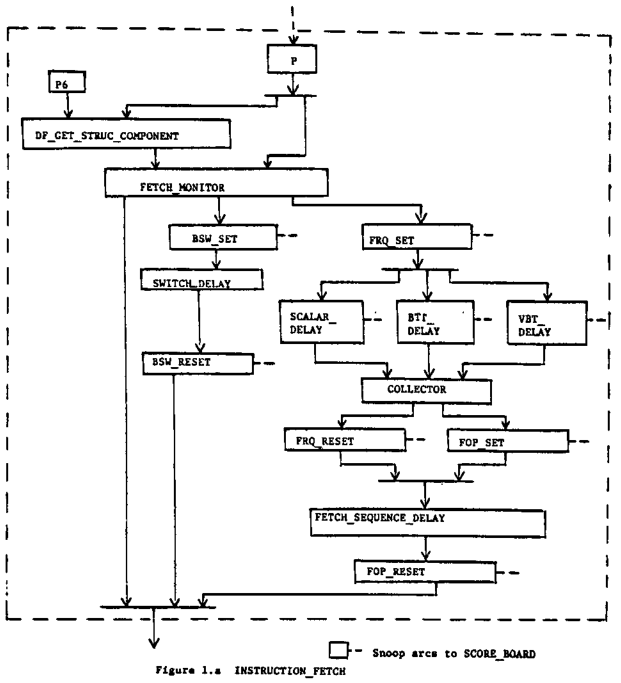

Figure 1.a INSTRUCTION_FETCH

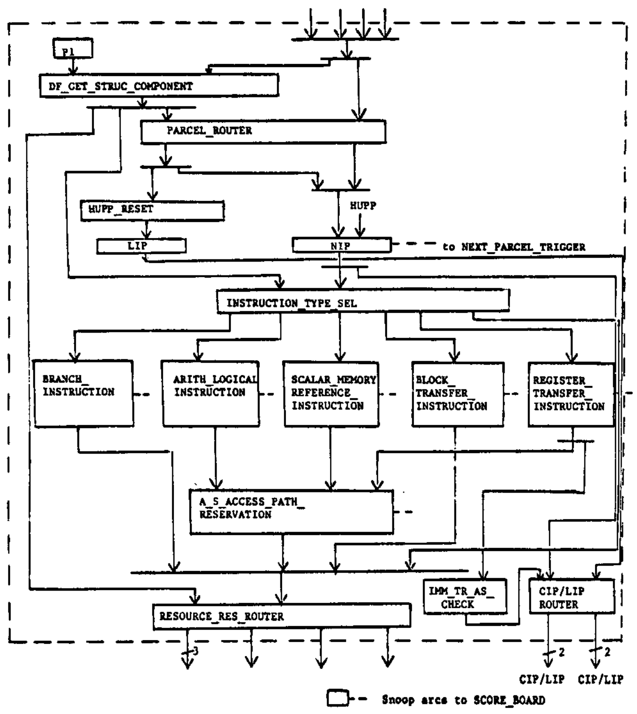

Figure 1.b INSTRUCTION_ISSUE_CHECKER

The INSTRUCTION_ISSUE_CHECKER is shown in Figure 1.b. It represents the instruction registers NIP, LIP, CIP, the instruction decoding logic, and the necessary logic to check the issue conditions of instructions against the status of the resources. To check the issue conditions, the instructions of the Cray-1S, except the monitor instructions and I/O instructions, are classified into one of the following types:

a. Branch
b. Arithmetic and logical
c. Scalar memory reference
d. Block transfer
e. Register transfer

For each type, a node is provided in the INSTRUCTION_ISSUE_CHECKER to check the issue conditions. These five nodes communicate with the SCORE_BOARD to obtain the status of the resources. The node latency time of INSTRUCTION_ISSUE_CHECKER varies depending on the instruction type processed and the resources available. The node latency time is provided by the SCORE_BOARD.

To reserve registers, memory access network and a functional unit, the INSTRUCTION_ISSUE_CHECKER outputs tokens on one of the first four sets of output arcs. The four sets of output arcs go into RESULT_OPERAND_RESERVATIONS, MEMORY-ACCESS_NETWORK, BLOCK_TRANSFER_ISSUE, and BRANCH INSTRUCTION_ISSUE nodes respectively. Two or more instructions using an A-register (S-register) as a result register cannot complete their execution at the same time. This is due to a single access path to A-register (S-register) group. So during a clock period at the most one A-register (S-register) can receive data. The A_S_ACCESS PATH_RESERVATION node detects A-register or S-register access path conflicts by communicating with the SCORE_BOARD and delays the issue of instructions that have conflicts.

Instructions in Cray-1S can be two parcels long. The upper parcel is sent to NIP, the lower parcel is sent to LIP, and a blank parcel is sent to the NIP corresponding to the lower parcel. This situation is accurately modeled in Figure 1.b.

The RESULT_OPERAND_RESERVATIONS unit is shown in Figure 1.c. It models the mechanisms in Cray-1S that reserves result registers and operand registers for vector instructions, and destination register for register transfer and scalar memory reference (read) instructions. There is a high degree of concurrency in the Cray-1S mechanisms performing the register reservations. This is modeled by the various parallel paths in Figure 1.c.

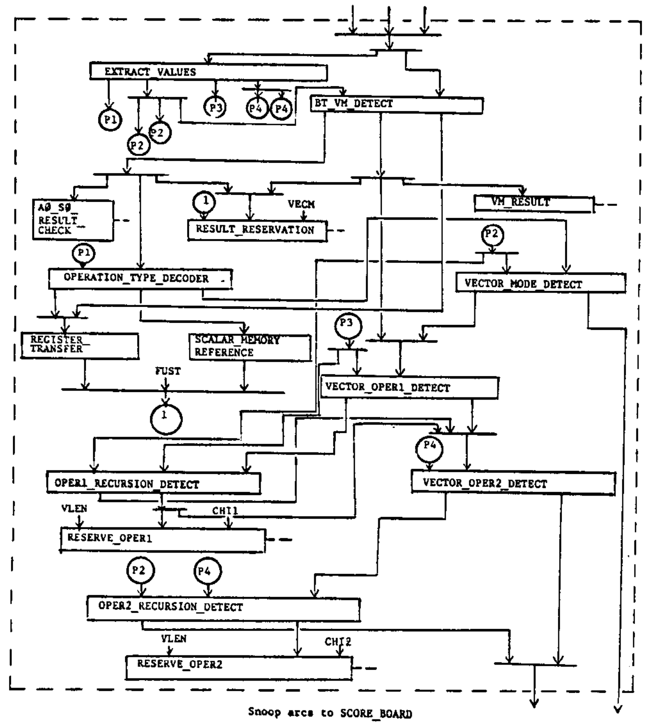

Figure 1.c RESULT_OPERAND_RESERVATIONS

Instructions of the arithmetic and logical, scalar memory reference, and register transfer types use registers. If an instruction of the above type is issued, the mechanisms in Cray-1S that reserve registers are notified. This situation is represented by the arrival of a token on one of the first three input arcs in Figure 1.C. Reserving result registers in the A-register, S-register, VL-register, or vector register group is carried out by the first path emanating from the BT_VM_DETECT node. Reserving the vector mask (VM) register as a result register is carried out by the second path emanating from the BT_VM_DETECT node. Transferring the contents of registers is carried out by the third path emanating from the BT_VM_DETECT node.

The mechanism for setting and synchronizing the A0-register busy (A0B) and S0-register busy (S0B) whenever one of these registers is used as a result register, is represented by the A0_S0_RESULT_CHECK node. The mechanism for reserving the result register of an instruction for the duration of the instruction execution time is represented by the RESULT_RESERVATION node. For register transfer and scalar memory reference instructions, the result register reservation time is supplied by the nodes REGISTER_TRANSFER and SCALAR_MEMORY_REFERENCE respectively. For arithmetic and logical instructions, the register reservation time is supplied by the FUNCTIONAL_UNIT_SEL/RES node (c.f. Figure 1.g).

In Cray-1S, operand registers and functional units of scalar arithmetic instructions are not reserved. But, operand registers and the functional units used by the vector instructions have to be reserved. The mechanism for detecting vector arithmetic and logical instructions is represented by the VECTOR_MODE_DETECT node. A vector instruction can use S-registers or vector registers as source operands.

If the first operand is a S-register or a vector register which is the same as the result vector register, then no register reservation is required. If the first operand is a distinct vector register, then the register is reserved. These two situations are represented by the two paths emanating from the VECTOR_OPER1_DETECT node. In a similar way, the second operand is checked and a vector register is reserved if necessary. The second operand checking and reservation is represented by the VECTOR_OPER2_DETECT node.

The MEMORY_ACCESS_NETWORK, shown in Figure 1.d, represents the mechanisms in Cray-1S to write from S-registers or A-registers to memory. Since scalar memory references can be issued every 2 CPs, memory bank conflicts can arise if the references are to the same bank. The rank register mechanisms to detect and resolve bank conflicts are also represented in Figure 1.d.

Each scalar memory reference in Cray-1S contains the memory bank number for the reference. The bank number is checked against the contents of rank registers B and C and if it is the same, then a bank conflict is present. The above action is represented by the BANK_CONFLICT_DETECT node. The rank registers are represented by RANK_A, RANK_ B, and RANK_C nodes with a node latency time of one. A conflict in rank B holds the memory reference from entering rank A register by two CPs while a rank C conflict causes a one CP delay. The above situation is represented by the nodes RB_CONFLICT_DELAY and RC_CONFLICT_DELAY with node latency time of two and one respectively.

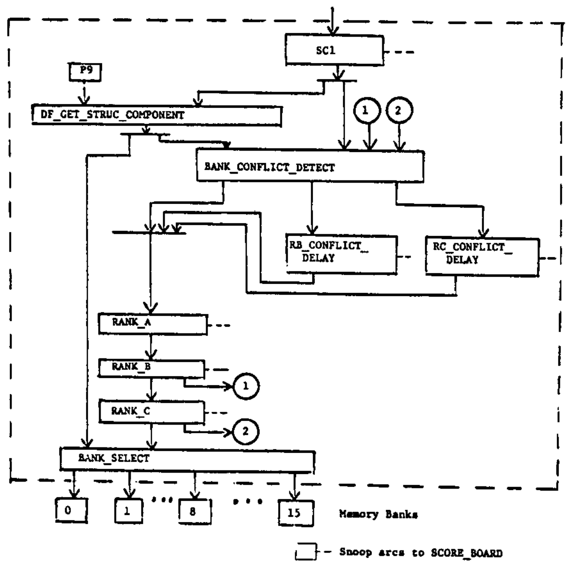

Figure 1.d MEMORY_ACCESS_NETWORK

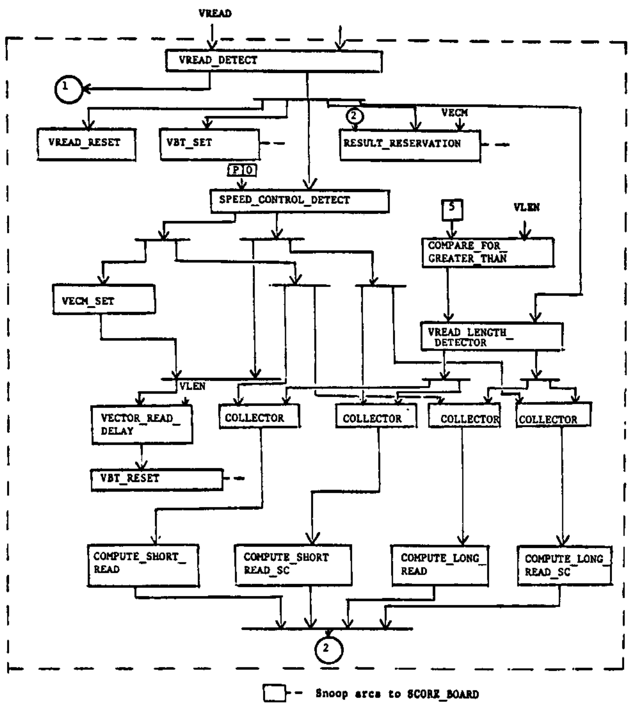

Figure 1.e BLOCK_TRANSFER_ISSUE (Read)

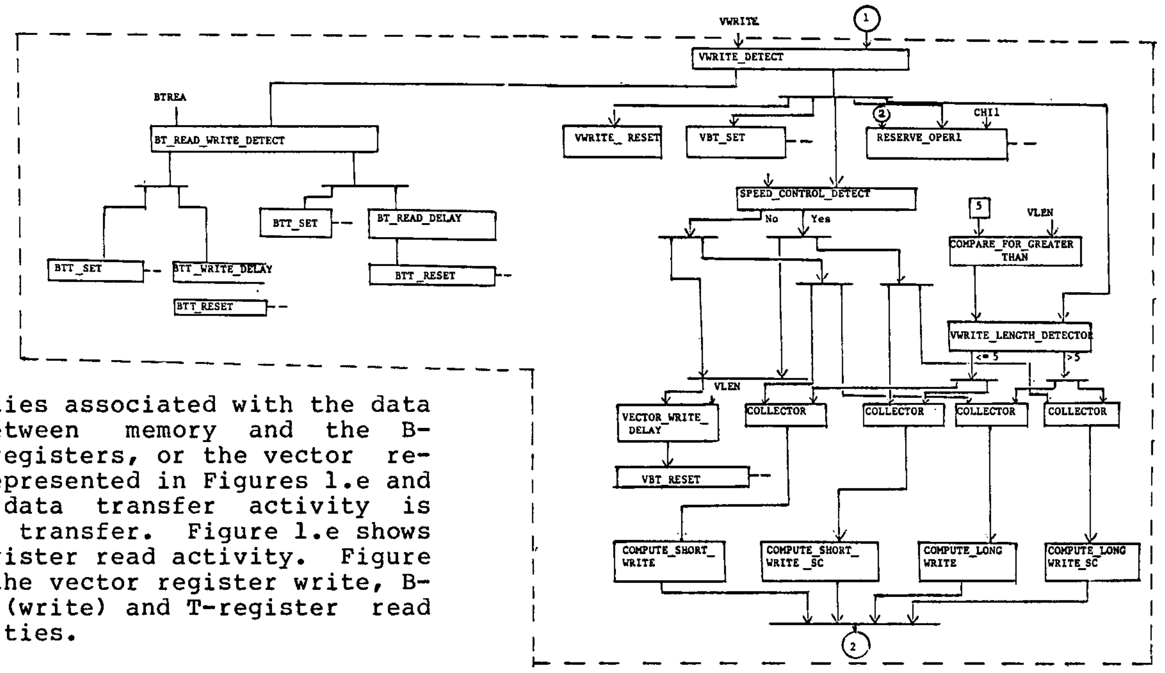

Figure 1.e.1 BLOCK_TRANSFER_ISSUE (Write)

The activities associated with the data transfers between memory and the B-registers, T-registers, or the vector registers are represented in Figures 1.e and 1.e.1. The data transfer activity is called block transfer. Figure 1.e shows the vector register read activity. Figure 1.e.1 shows the vector register write, B-register read (write) and T-register read (write) activities.

The VREAD_DETECT node detects block transfer to one of the vector registers and initiates four concurrent activities. The four activities are represented by four parallel paths. The first activity resets VREAD. The second activity sets the VBT flag and communicates the state change to the SCORE_BOARD. This is done to prevent the issue of another block transfer instruction while this transfer is in progress. The third activity computes the memory hold time and the time for which the result register has to be reserved (register reserve time). The fourth activity reserves the result register for the specified time.

The memory hold time and register reserve time depend on the vector length and the use of speed control. The details involved in the time calculations are shown in Figure 1.e.

The VWRITE_DETECT node determines whether the block transfer is a write into memory from vector registers or block transfers between memory and B_registers (T-registers). For a write into memory from vector registers, four concurrent activities are initiated. Each activity is represented by a path emanating from the VWRITE_DETECT node. The first activity resets VWRITE. The second activity sets VBT and communicates the state change to the SCORE_BOARD. The third activity computes the memory hold time and the time for which the first operand register is reserved. The fourth activity reserves the first operand register.

The block transfer between memory and B-registers (T-registers) involves making state change to BTT, communicating the state change to SCORE_BOARD, and computing the memory hold time. This is represented by the two paths emanating from the BT_READ_WRITE_DETECT node.

The BRANCH INSTRUCTION_ISSUE node, shown in Figure 1.f, represents the mechanism in Cray-1S to handle both conditional and unconditional jumps. The branch address of a jump instruction can be supplied in a B-register or as an immediate operand using two parcel instructions. For a jump instruction using a B-register, if the parcel following the jump instruction is not in the current buffer, but is in another buffer then the jump instruction execution time is increased by 2 CPs. Otherwise the execution time is increased by 11 CPs. This is represented by the B_JUMP_DELAY node where the node latency time is either two or eleven.

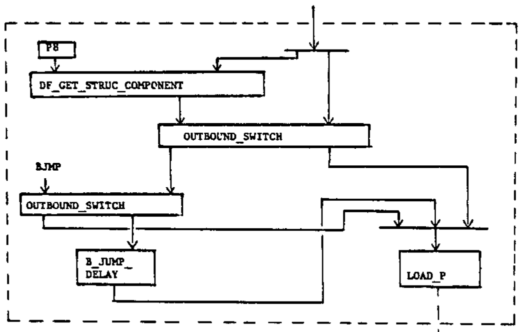

Figure 1.f BRANCH_INSTRUCTION_ISSUE

The FUNCTIONAL_UNIT_SEL/RESERVE unit, shown in Figure 1.g, represents the selection of a functional unit for address and scalar instructions. It also represents the selection and reservation of a functional unit for vector instructions.

SCORE BOARD

The system resources in Cray-1S are acquired, used, and released by the various components of Cray-1S. The status of the resources are kept in the system and it is available to all the components of the system. The state information of the resources in Cray-1S is represented using the dataflow graph SCORE_BOARD, shown in Figure 1.h. For each class of resource in Cray-1S, there is a node in SCORE_BOARD containing the state information. Snoop arcs are used in sending state information to the nodes in the INSTRUCTION_ISSUE UNIT and receiving state change information from these nodes. The mnemonics used in Figure 1.h are explained in Table 1.

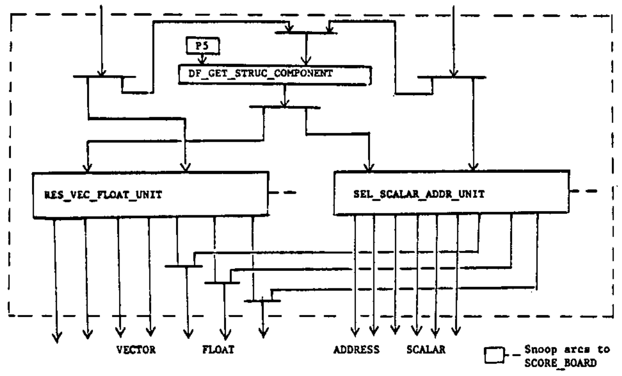

Figure 1.g FUNCTIONAL_UNIT_SEL/RES

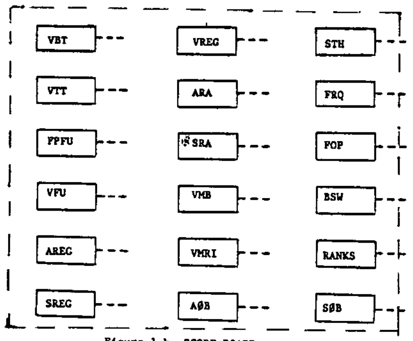

Figure 1.h SCORE_BOARD

Table 1

Mnemonic	Description
A0B	Register A0 busy
ARA	A-register access path
AREG	A-registers
BSW	Buffer switch
BIT	B-register or T-register transfer
CIP	Current instruction parcel
FOP	Fetch Operation
FPFU	Floating point functional units
FRQ	Fetch request
LIP	Lower instruction parcel
M	Memory
NIP	Next instruction parcel
P	Program counter
S0B	Register S0 busy
SRA	S-register access path
SREG	S-registers
STH	Storage hold
VBT	Vector block transfer
VECM	Vector mode flag
VFU	Vector functional units
VLEN	Vector length
VMB	Vector mask busy
VMRI	Vector mask read inhibit
VREG	V-registers

ADDRESS UNIT

This unit represents 8 A-registers, 64 B-registers, two functional units for address calculations, and a controller that controls among other things the A-register input path so that one A-register is loaded at a time. A dataflow graph of the unit is shown in Figure 2.

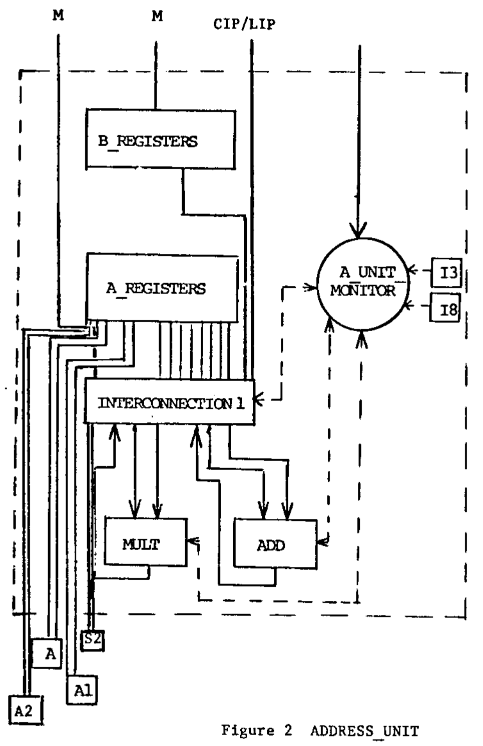

Figure 2 ADDRESS_UNIT

The node INTERCONNECTION1 represents the A-register input path and output path. Input to A-registers can come from B-registers, memory, functional units ADD and MULTIPLY, CIP/LIP, and from S-registers. These inputs are represented by the various input arcs and the undirected arcs to INTERCONNECTION1. The undirected arcs represent paths that can be used either for input or output. At the most one A-register can be receiving an input from one of the above mentioned sources during a clock period (CP). The contents of A-registers are available simultaneously to the above mentioned sources. The node A_UNIT_MONITOR represents the local controller for the ADDRESS_UNIT. The A_UNIT_MONITOR specifies the control information to INTERCONNECTION1, INTERCONNECTION3 in Figure 3, and INTERCONNECTION8 in Figure 5 so that data transfer can take place between the A-registers and the various functional units. It also provides the operation that a functional unit has to do and facilitates the release of result A-registers when address computations are completed.

The ADD node represents the pipelined add functional unit and has a node latency time of 2 CPs. The MULTIPLY node represents the pipelined multiply functional unit and has a node latency time of 6 CPs.

SCALAR UNIT

This unit represents 8 S-registers, 64 T-registers, four functional units capable of performing arithmetic and logical operations on scalars, and the local controller of the functional units and the registers. A dataflow graph of the unit is shown in Figure 3.

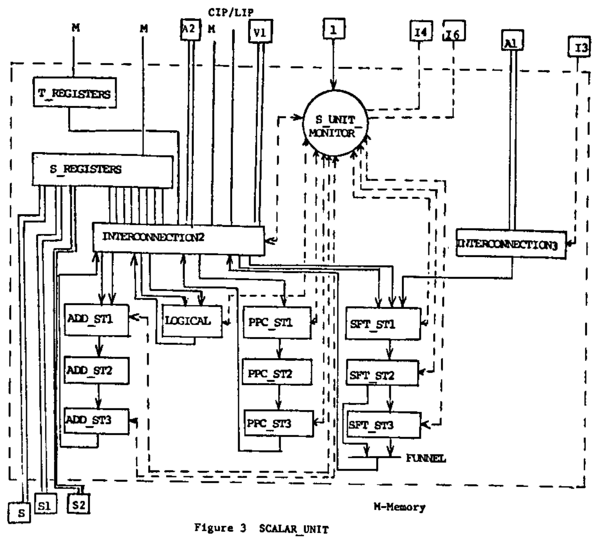

Figure 3 SCALAR_UNIT

The node INTERCONNECTION2 represents the S-register input path and output path. Input to S-registers come from T-registers, memory, one element of a vector register, the four scalar functional units, or CIP/LIP. These inputs are represented by the various input arcs and the undirected arcs to INTERCONNECTION2. At the most one S-register can receive an input from the above sources during any CP. The node INTERCONNECTION3 represents the input path from A-registers to the shift unit. INTERCONNECTION2 gets the control information from the local controller, S_UNIT_MONITOR. The controller provides the operation that a functional unit has to do, maintains the S-registers, obtains the status of the functional units, and facilitates the release of result S-registers when instructions have completed their executions.

The scalar functional units are pipelined. Each stage of a pipelined functional unit is represented by a node in Figure 3 and the node latency time of each of these nodes is one CP. The last stages of the pipelines put the results in the S-registers and communicate to the S_UNIT_MONITOR the successful transfer of results to the S_registers.

FLOAT UNIT

This unit represents three pipelined floating point functional units, interconnection networks for making connections between the functional units and S-registers and vector registers, and the local controller of the functional units and the interconnection networks. A dataflow graph of the unit is shown in Figure 4. The node INTERCONNECTION4 represents the interconnection network between the S-registers and the floating point functional units. The control information for the interconnection network is supplied by the S_UNIT_MONITOR in Figure 3. The node INTERCONNECTION5 represents the interconnection network between the V-registers and the floating point functional units. The control information for the interconnection network is supplied by the V_UNIT_MONITOR in Figure 5.

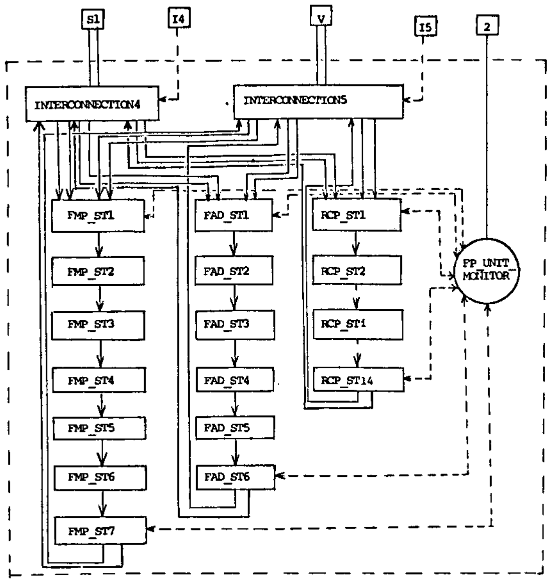

Figure 4 FLOAT_UNIT

The pipelined floating point multiplier with seven stages is represented by seven nodes, FMP_ST1 through FMP_ST7. Each of the seven nodes has a latency time of one CP. The last stage of the pipeline, FMP_ST7, puts the result in a S-register (vector register) and communicates to the FP_UNIT_MONITOR the successful transfer of results. The pipelined floating point adder with six stages is represented by six nodes, FAD_ST1 through FAD_ST6. Each of the six nodes has a latency time of one CP. The last stage of the pipeline, FAD_ST6, puts the result in a S-register (vector register) and communicates to the FP_UNIT_MONITOR the successful transfer of results. The pipelined floating point divider using reciprocal approximation is represented by 14 nodes, RCP_ST1 through RCP_ST14. Each of the 14 nodes has a latency time of one CP.

The node FP_UNIT_MONITOR provides the operation that a functional unit must do, and facilitates the release of registers after the completion of operations.

VECTOR UNIT

This unit represents 8 vector registers, four functional units capable of performing fixed point arithmetic, logical, shift, and population count operations, interconnection networks between the functional units and vector registers, S-registers and A-registers, and the local controller of the functional units and the interconnection networks. A dataflow graph of the unit is shown in Figure 5.

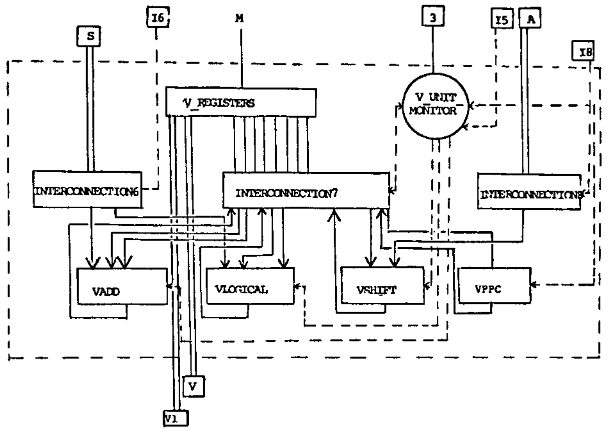

Figure 5 VECTOR_UNIT

----- Snoop arc bundle
___ Arc
═══ Arcbundle (8 arcs)
—[x] [x]— Link between two ends of an arc

The node INTERCONNECTION6 represents the interconnection network between the S-registers and the vector functional units. The control information for the interconnection network is supplied by the S_UNIT_MONITOR in Figure 3. The node INTERCONNECTION7 represents the interconnection network between the vector registers and the vector functional units. The control information for the interconnection network is supplied by the V_UNIT_MONITOR. The interconnection between A-registers and the vector shift unit is represented by INTERCONNECTION8. The four pipelined vector functional units are represented by the nodes VADD, VLOGICAL, VSHIFT, and VPPC. They have the functional unit node latency times of 3, 2, 4, and 4 respectively. The node V_UNIT_MONITOR provides the operation that a functional unit has to do, communicates the control information to interconnection networks, and facilitates the release of registers after the completion of instruction executions.

A complete description of the nodes in the dataflow graph model of Cray-1S as well as the GPSS blocks used to implement the nodes are in [16].

4 EXPERIMENTS WITH THE CRAY-1S MODEL

In this section, the results obtained from running several benchmark programs on the CRAY-1S model are presented. The simulated execution times of some of these programs are compared with the actual execution times on a CRAY-1S computer. In addition, the results of some experiments performed on the model are included. These experiments are aimed at studying the effect of removing some of the bottlenecks in the CRAY-1S architecture. With each experiment, the simulated execution time and the rate at which instructions are issued are presented. Both these statistics are given in simulated clock periods (12.5 nanoseconds). Finally, some areas of future work are discussed including some additional enhancements to the architecture.

Programs Used in the Analysis

Control streams for four sample programs are generated and used as input in the simulation runs. The first program is a full scalar matrix multiplication called FSMTRXM. This program performs matrix multiplication by using scalar instructions exclusively (i.e., there are no vector operations involved). The second program, called FVMTRXM, uses mostly vector instructions to perform the matrix multiplication. Both matrix multiplication programs along with the execution times on CRAY-1S were supplied by Cray Research, Inc. The other programs used, called CSPLIN and TRIMAT, are portions of subroutines employed in the feasibility study of FMP [4,18].

Table 2 shows the distribution of instruction types for each of the four programs. Note that the arithmetic-logical type has been broken down into scalar and vector instructions. The time taken to complete the simulation for each program on the model without changes is shown in Table 3. For FSMTRXM and FVMTRXM the actual execution times on CRAY-1S for one iteration (multiplication of two scalar values) and five iterations (multiplication of two 5 x 5 matrices) are also shown. In each case, the simulated execution time is within acceptable bounds of the actual execution time. For the purpose of the experiments, the five iteration versions of FSMTRXM and FVMTRXM are used. Also shown on this table is the average time in CPs an instruction parcel spent in CIP waiting to issue. This is an important statistic in measuring performance since it represents the rate at which instructions are issued. Under ideal conditions, this time should be 1 CP. The average time per parcel in NIP is also shown.

In addition to the four programs described above, a number of small programs have been used to determine the accuracy of the model. These programs tested operations such as branches and buffer fetch sequences. In all these cases, the simulated times agreed with the time taken to run the programs on CRAY-1S.

The dataflow model of Cray-1S is analyzed to detect potential bottlenecks in the CRAY-1S architecture. Several clues

Table 2 Introduction Type Distribution In Programs

Type of Instruction	FSMTRXM	FVMTRXM	CSPLIN	TRIMAT
Branch	7.63	5.01	2.68	3.44
Arithmetic-logical				
Scalar	41.47	33.64	28.76	47.13
Vector	0.00	2.84	12.04	0.00
Scalar Memory Reference	38.48	19.59	7.02	21.31
Block Memory Transfer	0.10	4.34	13.71	0.16
Register Transfer	12.32	34.57	35.79	27.95

Table 3 Statistics For Model Without Changes

Program	CRAY-1S Execution Time	Simulated Time	Average Time Per Parcel In NIP	Average Time Per Parcel In CIP
FSMTRXM (1 iteration)	269	269	1.88	1.79
FSMTRXM (5 iterations)	10665	10625	1.62	1.58
FVMTRXM (1 iteration)	364	361	1.89	1.92
FVMTRXM (5 iterations)	4824	4681	1.84	1.85
CSPLIN	--	3061	8.92	8.92
TRIMAT	--	3426	2.15	2.12

for improving the performance have been found. Based on these clues, a list of changes that can be readily implemented on the model is found. In the following sections, these changes and the results obtained when they are simulated on the model are presented.

Experiment 1: Changing the Number and Size of Instruction Buffers

This experiment consists of reducing the number of instruction buffers in the model from four to two while increasing the size of each buffer from 16 words to 32 words. This means that the look-ahead capability remains the same (256 parcels). The change is simulated for transfer rates of four words per CP (as in the model without changes) as well as eight words per CP.

The results in Table 4 show a slight decrease in simulated execution time for the programs. The decrease is not very significant due to the relatively small size of the programs. This allows the processor to operate in loop-mode once the buffers are loaded.

Table 4 Results from Experiment 1

4 words per CP

Program	Simulated Time	Percent Improvement	Average Time per Parcel In CIP	Percent Improvement
FSMTRXM	10609	0.15	1.58	--
FVMTRXM	4625	1.20	1.85	--
CSPLIN	2991	2.29	8.78	1.61
TRIMAT	3422	0.12	2.12	--

8 words per CP

Simulated Time	Percent Improvement	Average Time per Parcel In CIP	Percent Improvement
10605	0.19	1.58	--
4621	1.28	1.85	--
2983	2.55	8.78	1.61
3414	0.35	2.12	--

Experiment 2: Elimination of Single A-S Access Path

This experiment is designed to study the effect of removing the constraint of a single access path to A registers and S registers. This corresponds to removing the A_S_ACCESS_PATH_RESERVATION node from the dataflow model. In this way, any number of A registers and S registers can be entered with information at a given clock period.

The results of this experiment are shown in Table 5. A slight decrease in the simulated execution time is obtained for all four programs. The average time spent by an instruction in CIP decreases in proportion to the simulated execution time. These results tend to indicate that the single access path to A registers (S registers) is not a significant bottleneck in the CRAY-1S architecture. This is true even for programs which use scalar instructions almost exclusively like FSMTRXM and TRIMAT.

Table 5 Results From Experiment 2

Program	Simulated Time	Percent Improvement	Average Time Per Parcel In CIP	Percent Improvement
FSMTRXM	10600	0.24	1.57	0.25
FVMTRXM	4646	0.75	1.83	0.76
CSPLIN	3057	0.13	8.91	0.13
TRIMAT	3405	0.61	2.10	0.66

Experiment 3: Reduced Memory Bank Cycle Time

The next experiment consists of reducing the memory bank cycle time in half (from 4 CPs to 2 CPs). This reduction has the effect of eliminating the possibility of bank conflicts for scalar memory references. This is due to the fact that scalar references take 2 CPs to issue. In addition, the reduced bank cycle time decreases the possibility of bank conflicts in vector block transfers. Speed control [2] is now required only when the starting memory address is incremented by a multiple of 16. In this case, the transfer rate is one word every 2 CPs.

The results in Table 6 indicate that programs like FSMTRXM, FVMTRXM, and TRIMAT which have a large number of scalar memory references, benefit significantly from this change. The decrease in execution time for CSPLIN was very slight. This is as expected since this program has a large number of vector block transfers and the vector lengths are relatively long (around 150 elements). A reduction in bank cycle time only speeds up the setup time for vector block transfer operations.

Table 6 Results From Experiment 3

Program	Simulated Time	Percent Improvement	Average time Per Parcel In CIP	Percent Improvement
FSMTRXM	10017	5.72	1.48	6.22
FVMTRXM	4389	6.24	1.73	6.34
CSPLIN	3036	0.82	8.88	0.44
TRIMAT	3225	5.87	1.99	6.00

Experiment 4: Increased Memory Bandwidth for Vector Transfers

The results of experiment 3 indicate that a faster bank cycle time has no significant effect on vector block transfers. In the next experiment, the memory bandwidth for vector block transfers is increased so that up to four memory banks can be accessed simultaneously. This allows a transfer rate of up to four words per CP when successive references to a particular bank are at least 4 CPs apart.

The results in Table 7 show a 30.97% decrease in the simulated execution time for CSPLIN and a 31.45% decrease in the average time an instruction spends in the CIP register. The decrease is larger than expected even for a program with a high number of vector block transfers. This is probably due to the comparatively large size of the vectors in this program. This change had no effect on the other programs. For FSMTRXM and TRIMAT this is to be expected since no vector block transfers are involved. In the case of FVMTRXM, the vector lengths are of only five elements. This means that any increase in the transfer rate is going to be offset by the vector startup time.

Table 7 Results From Experiment 4

Program	Simulated Time	Percent Improvement	Average time Per Parcel In CIP	Percent Improvement
FSMTRXM	10625	-	1.58	-
FVMTRXM	4681	-	1.85	-
CSPLIN	2113	30.97	6.11	31.45
TRIMAT	3426	-	2.12	-

Experiment 5: Extension of Chain Slot Time

Chaining of two vector operations [1,2] can take place only at chain slot time (i.e., functional unit time plus 2 CPs). If the second operation is not ready to issue at this time, then it has to wait for the completion of the first instruction. The performance of the system can be further improved by allowing chaining to take place at any time between functional unit time plus 2 CPs and the end of a vector operation. This experiment consists of implementing such a change in the model. The results of this experiment are shown in Table 8. The execution time of CSPLIN decreased by 8.92% while the average time per parcel in CIP decreased by 9.19%. This is as expected since there are several successive vector instructions in this program, each processing relatively long vectors. Extending the chain slot time should benefit these operations since missing the opportunity to chain means a considerable wait for the release of a result register. This wait in turn holds the entire instruction pipeline. Surprisingly, the change had no effect on FVMTRXM. This is probably due to the shorter vector lengths in this program and to the distribution of the vector instructions (i.e., they are not clustered together).

Table 8 Results From Experiment 5

Program	Simulated Time	Percent Improvement	Average time Per Parcel In CIP	Percent Improvement
FSMTRXM	10625	--	1.58	--
FVMTRXM	4681	--	1.85	--
CSPLIN	2788	8.92	8.10	9.19
TRIMAT	3426	--	2.12	--

Experiment 6: Parallel Instruction Issue

The purpose of the experiments described so far has been to speed up the execution of certain operations such as memory transfers and to increase the rate at which instructions are issued by eliminating certain issue conditions such as the single access path to scalar registers. The fact still remains that the INSTRUCTION_ISSUE_CHECKER services instructions on a first-in-first-out basis. This means that when an instruction cannot be issued because some condition is not satisfied, subsequent instructions cannot be issued either. In many cases, these instructions are ready to issue and can potentially even complete their execution before the one in CIP issues. This is not possible under the present system because only one CIP/LIP register pair is available. In this experiment, the model is modified to include two CIP/LIP register pairs. This increases the bandwidth of the INSTRUCTION_ISSUE_CHECKER by allowing up to two instructions to issue at every clock period. In addition, instead of servicing instructions on a strictly first-in-first-out basis, an instruction can be issued ahead of its predecessor in the pipeline, provided no procedural conflicts [10] exist between the instructions. When an instruction is in one CIP and a second instruction is ready to move to another CIP, the following conditions are checked:

a. The results and operand registers of the second instruction must be different from the result register of the first instruction.
b. The result register of the second instruction must be different from the operand(s) of the first instruction.
c. If the second instruction is a vector instruction, its operand(s) must be different from the operand(s) of the first instruction.
d. If the first instruction is a vector instruction, the second instruction cannot load the VL register.
e. If the first instruction references memory, the second instruction cannot.
f. The functional units required by the instructions must be different.

If any of these conditions are not satisfied, the second instruction is kept in NIP until the first instruction is issued.

Some of these conditions are strictly procedural and cannot be avoided. However, there are some which are included to simplify the changes to the model. This means that whatever results are obtained can be further improved if some additional changes are made.

The results of this experiment are shown in Table 9. As can be seen, even this restricted scheme of parallel instruction issue had a significant effect on all four programs. This confirms our assumption that the single set of instruction registers presented a major bottleneck. For this experiment, the average time spent by an instruction parcel in NIP is given. This time is then compared with the time obtained in the model without changes. For all programs, a significant increase in the rate at which instruction parcels leave this register can be observed. The average time each parcel spent in CIP remained the same as in the model without changes. This is to be expected since the extra CIP/LIP register pair has no effect on the issue conditions of individual instructions.

Table 9 Results From Experiment 6

Program	Simulated Time	Percent Improvement	Average Time Per Parcel In NIP	Percent Improvement
FSMTRXM	9859	7.21	1.47	8.90
FVMTRXM	3886	16.98	1.50	18.45
CSPLIN	2957	3.40	8.46	5.19
TRIMAT	3182	7.12	1.99	7.35

5 CONCLUSIONS

The research effort considered the effects of removing several potential bottlenecks in the CRAY-1S on its performance. The limited number of experiments conducted on the model shows that there are some areas in which the architecture can be improved.

In terms of the main memory unit, two significant improvements can be made. For programs with predominantly scalar memory references, a reduction in the bank cycle time contributes significantly to improve system performance. For programs with a considerable amount of vector memory instructions, a higher transfer rate can have a considerable effect.

Experiment 5 showed that for programs with a high concentration of vector instructions, an extended chain slot time contributes significantly to accelerate the rate at which instructions are issued.

Experiment 6 showed fairly conclusively that the INSTRUCTION_ISSUE_CHECKER is a major bottleneck in the CRAY-1S architecture. This experiment had a significant effect on all four programs. It showed that even with the restrictions imposed, an extra CIP/LIP register pair can improve system performance considerably.

A change that was not implemented in the model but which can improve the performance of vector operations is to treat the V registers as FIFO queues. Under this scheme, two pointers are maintained for each V register: one for the front of the queue and one for the back of the queue. In this way, only the register elements are reserved during the clock period in which a result is transferred to them from a functional unit.

An interesting area for future work is to extend the concept presented in experiment 6 to include multiple instruction units, each connected to an instruction buffer. With the present number of buffers, four independent instruction units can be implemented. These units can potentially operate in a tightly-coupled environment by using a single program counter, or by using several program counters operating in parallel.

ACKNOWLEDGEMENTS

We are thankful to Martin Storma of Cray Research Inc. for supplying some of the test programs and discussions on some of the intricacies of Cray-1S.

REFERENCES

1. R. M. Russell, "The Cray-1 Computer System", Communications of the ACM, Vol. 21, No. 1, Jan. 1978, pp 63-72.
2. Cray-1S Series Hardware Reference Manual, HR-0808, June 1980, Cray Research Inc., Mendota Heights, MN 55120.
3. E. W. Kozdrowicki, and D. J. Thesis, "Second Generation of Vector Supercomputers", Computer Magazine, Nov. 1980, pp 71-83.
4. S. F. Lundstrom, and G. H. Barnes, "A Controllable MIMD Architecture", Proceedings of the 1980 Conference on Parallel Processing, Michigan, Aug. 1980, pp 49-52.
5. D. E. Lang, "Modeling for Parallel-Pipeline Central Processors", Ph.D. Dissertation, 1979, University Microfilm, Ann Arbor, MI, No. 7920153.
6. J. M. Mirza, "Analysis and Design of Pipeline Processors", Ph.D. Dissertation, 1979, University Microfilm, Ann Arbor, MI, No. 7920792.
7. J. R. Enshoff, and R. L. Sisson, Design and Use of Computer Systems Models, Macmillan Publishing Co., New York, 1970.
8. J. B. Dennis, "First Version of a Data Flow Procedure Language", Project MAC Tech. Memo: 61, May 1975, MIT, Cambridge, MA.
9. V. P. Srini, "An Extended Abstract Dataflow Methodology for Designing and Modeling Reconfigurable Systems", Ph.D. Dissertation, 1980, University Microfilm, Ann Arbor, MI, No. 8100289.
10. J. L. Baer, Computer Systems Architecture, Computer Science Press Inc., Potomac, MD, 1980.
11. D. W. Anderson, F. J. Sparacio, and R. M. Tomasulo, "The IBM 360 Model 91: Machine Philosophy and Instruction Handling", IBM Journal of Research and Development, 11, Jan. 1967, pp 8-24.
12. J. E. Thornton, Design of a Computer System: The Control Data 6600, Scott, Foresman and Co., Glenview, IL, 1970.
13. G. Gordon, The Application of GPSS V to Discrete System Simulation, Prentice-Hall Inc., Englewood Cliffs, NJ, 1975.
14. IBM Corp., General Purpose Simulation System V Users Manual, SH20-0851-1, Aug. 1971, IBM, White Plains, NY 10604.
15. S. P. Landry, and B. D. Shriver, "A Simulation Environment for Performing Dataflow Research", 1979 Conference on Simulation, Measurement and Modeling of Computer Systems, Boulder, Colorado, Aug. 1979, pp 131-139.
16. J. F. Asenjo, Analysis of Cray-1S using Dataflow Graphs, M.S. Thesis, Computer Science Dept., Univ. of Alabama, Birmingham, AL 35294, June 1982.
17. B. Kumar, and E. S. Davidson, "Performance Evaluation of Highly Concurrent Computers by Deterministic Simulation", Communications of the ACM, Vol. 21, NO. 11, Nov. 1978, pp 904-913.
18. NASA Ames Research Center, "Numerical Aerodynamic Simulator Processing System Specification - Appendix III, Version 1.0", PC 320-02, NASA Ames Research Center, Moffett Field, CA, Sept. 1980.