The Fluid Dynamics Machine Architecture

TR88-056

December 1988

Mark C. Davis

The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175

The Fluid Dynamics Machine Architecture

Mark C. Davis

CB # 3175, Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175

December 1, 1987

Abstract

The Fluid Dynamics Machine(FDM) is designed to rapidly solve an important physical problem. Fluid dynamics has application in automotive engineering, aeronautics and weather prediction. Previous computers and solution methods have been too slow or too expensive to be widely usable. This paper describes the architecture of the Fluid Dynamics Machine, a fine grain, systolic multiprocessor computer. The FDM has 9612 identical processing elements in a 96 by 96 array. Multiple communication paths between neighbors embed the processing elements in a logical hexagonal grid. Each of the processing elements simulates fluid flow using a cellular automata algorithm developed by Frisch. The processing elements also accumulate momentum average data to condense the quantity of output from the machine. A separate input-output processor controls the processing element array and communicates with a host workstation. Parameter selection to optimize a wafer scale implementation is also described. An implementation on 9 wafers, each 5 inches in diameter and using 1.2 micron design rules, could be built on one VME board and could calculate fluid flow faster than a modern workstation could display the results.

Contents

List of Figures

1 Preface

The Fluid Dynamics Machine (FDM) efficiently simulates the motion of fluids like air or water. A special purpose computer, FDM easily and economically implements Frisch's algorithm to solve the fluid dynamics problem. The FDM contains 9216 individual processing elements, putting it in the class of large order multiprocessor machines. The architecture is particularly suited to wafer scale implementation. The important problem, the efficient algorithm, the high performance multiprocessor architecture, and the economical wafer scale implementation makes this machine a valuable addition to the family of computing machinery.

I assume that the reader of this technical report already understands different types of multiprocessor architectures and interconnection schemes. Also the reader is familiar with Very Large Scale Integration (VLSI) design techniques and the concepts and difficulties of wafer scale integration. The VLSI and multiprocessor concepts are covered in Mead and Conway's introduction to VLSI systems[6]. The wafer scale considerations may be found in Hedlund[4].

This technical report describes the architecture of the fluid dynamics machine and some of the design considerations. The report begins with a discussion of the fluid dynamics problem, its applications and its solutions. A description of the architecture of the Fluid Dynamics Machine follows, including its major components of inter-processing element communications, processing element architecture and input-output processor architecture. The final section presents wafer scale considerations because the architecture is targeted for that wafer scale implementation.

2 Fluid Dynamics

Fluid dynamics is the study of the motion of a fluid like air or water. Moving fluids influence the activity of many engineering disciplines. Previous methods of predicting the behavior of fluid in motion have been very time consuming. The fluid dynamics machine will economically and rapidly predict the behavior of fluids in motion by using modern multiprocessor techniques and an algorithm particularly suited to this type of implementation. This section presents applications for fluid dynamics, current solutions and a features of the Fluid Dynamics Machine.

2.1 Many Fields Have Fluid Dynamics Problems

Many scientific disciplines require the solution of the fluid dynamics problem. When discussing fluid dynamics, the term fluid means gasses as well as liquids. Fluid dynamics is so complex that any nontrivial problem is very difficult to solve. Building a car, designing an airplane, or predicting the weather all require much prediction of fluid behavior. These three typical applications of fluid dynamics will be discussed.

2.1.1 Automotive Design

Automotive Engineers must determine the effects of moving fluids in several areas. The best size of power plant depends heavily on the wind resistance of the car's body. The volume of air flowing into a cylinder during an internal combustion engine cycle greatly effects the horsepower. The size of the water pump will be determined by the resistance to the flow of cooling water inside the engine block and radiator. All of these relationships can be calculated by solving the fluid dynamics problem for the specific case. The alternate method of building prototypes to measure these properties is very expensive and time consuming.

2.1.2 Aerodynamics

Perhaps the greatest present demand for fluid dynamics solutions is in the aircraft industry. In the early days of aircraft design, wings were designed by trial and error and models were made and tested before building a complete airplane. The modeling process became more sophisticated with the introduction of wind tunnels. Unfortunately, building models and running wind tunnel tests are very expensive activities. To get faster results and reduce costs, computers have been used to simulate airflow across the surfaces of an airplane. Although computers are used as much as feasible, the accuracy of these computations have been hampered by the large number of calculations required. Engineers accept conceptual models that simulate only laminar flow or uncompressible fluids because these models require fewer computations. As a result, extensive wind tunnel work is still required and the wind tunnel results can contain some unexpected observations.

2.1.3 Weather Prediction

Simulating the fluid dynamics of weather is a complicated problem, and it is so hard that solving it with conventional general purpose computers is infeasible. The volume of air involved and the numerous sources make this a very large problem. Also, the results are perishable: nobody wants a prediction of yesterday's weather. As a consequence, weather prediction is done on a very coarse scale and is notoriously inaccurate.

2.2 Navier-Stokes and Super Computers Can be Used

Although many people need solutions to the fluid dynamics problems, the combination of algorithms and computer systems used today are too slow and expensive. Today the most popular way to solve fluid dynamics problems accurately is to solve the Navier-Stokes equations. These differential equations are difficult to solve, and the only effective way is to run standard differential equation solving techniques on a supercomputer. Even with a supercomputer, obtaining results normally takes many hours. The slow turn around and the very high cost of these calculations cause great inconvenience for the few engineers who have the opportunity to use these expensive machines.

2.3 The FDM Solution

Since general purpose computer architectures inefficiently solve the fluid dynamics problem, scientists and engineers need a better approach. Our solution is a special purpose architecture, using an algorithm particularly suited for special purpose computers and having advanced multiprocessing features. The algorithm will be described, followed by the high performance techniques used in the FDM architecture: multiprocessing, large order parallelism, a systolic array and VLSI.

2.3.1 A Good Algorithm

Frisch[1] developed an algorithm to solve the 2 dimensional fluid dynamics problem. Working at the National Laboratory in Los Alamos, he found cellular automata produced accurate results. A cellular automata is a simple device that processes a few binary inputs to a few outputs. This technique correctly simulated compressible, non-laminar flow, the most difficult fluid dynamics problem. In his algorithm, the area containing the fluid is divided up into cells. The motion of the fluid is simulated by "particles" that may travel from cell to cell or remain stationary in the cell. Each cell is connected to its immediate neighbors by six paths. The location and direction of motion of all the particles traveling through the cells make up the state of the area under analysis. Figure 1, taken from d'Humieres [3], shows 24 cells with the connecting the paths.

When two particles attempt to occupy the same cell a set of collision rules determines the final placement of each particle. In some collisions, only one outcome is possible. Frequently, two outcomes are equally likely. In this case, a random choice of possible placements is made. One important set of rules describes open space; other sets describe boundaries, walls, sources, or sinks. Figure 2 shows rules for open space, with the configuration before the collision on the left and the possible results of the collision on the right. A wall oriented vertically requires different rules than a wall oriented in some other way.

If this model uses 1 million to 10 million cells to describe an area, it produces very good results. At this level of resolution, the Frisch algorithm will accurately predict the behavior of a wing in a wind tunnel. The algorithm performs much better than direct solution of the Navier-Stokes equations because the simple calculations to determine the rules can be executed much faster than the floating point operations required for direct solution. Also, since all computations are applications of simple rules using small data groups (seven particles represented as bits), it is well suited for implementation on a very small processing element.

2.3.2 Multiprocessors

In order to get higher performance, a computer may consist of multiple processing elements. As long as the problem can be partitioned into many parts, each processing element busily works on a separate part. Multiprocessor computers are not widely used today because few single problems adapt well to the partitioning required for multiprocessing. Fortunately, implementations of the Frisch algorithm are naturally partitioned to take most advantage of multiprocessing.

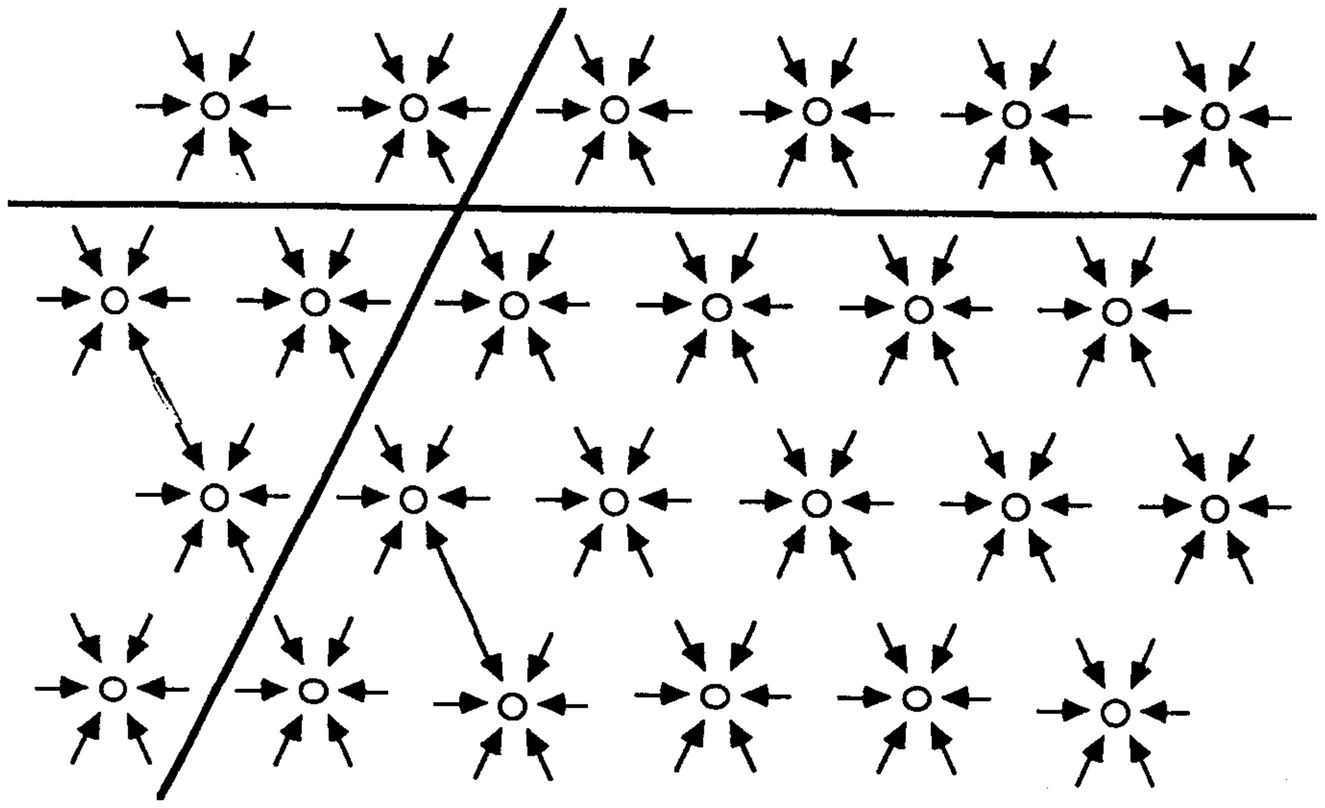


Figure 1: Cellular Automata Hexagonal Grid

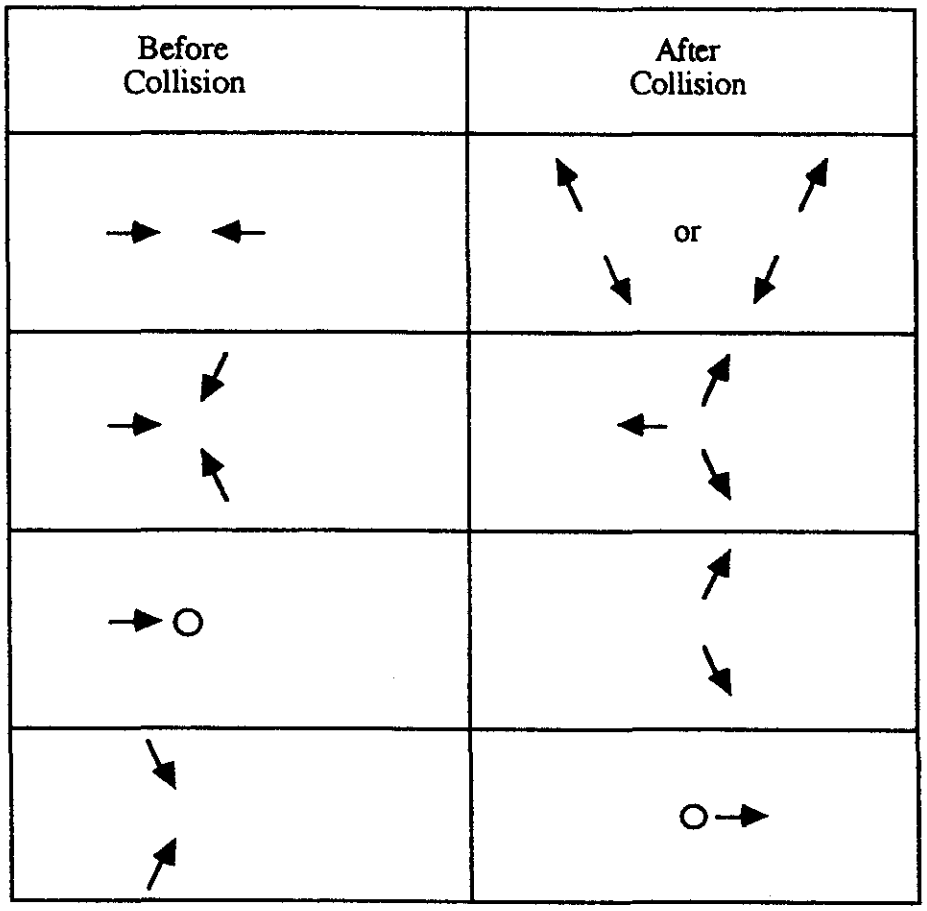


Figure 2: Open Space Collision Rules

2.3.3 Large Order Parallelism

Although some tasks permit some form of parallelism in processing, the maximum number of concurrent tasks is often limited to 10 or 20. The number of independent processes and the amount of shared data limit the number of processors that can effectively work on a problem simultaneously. Shared data reduces the available parallelism because standard memory can only be read and written in a serial fashion. Duplicating the memory and other tricks can improve access to the memory, but such techniques are expensive and will only improve the memory performance by an order of magnitude. With Frisch's algorithm, the partitioning of the problem and the character of the small amount of required data sharing permit thousands or millions of independent tasks.

2.3.4 Systolic Data Sharing

A small portion of the data must be shared between processes. This data may be passed from a processor to its neighbor. This movement of data is called systolic because it resembles the pumping of blood. The architecture of FDM is systolic, and as a result each processor needs to talk to only its neighbors, so no complicated, unreliable, or slow global data communication paths are required.

2.3.5 VLSI - Much Computing Power for Low Cost

The FDM architecture and the simplicity of the Frisch algorithm permit easy implementation using Very Large Scale Integrated (VLSI) circuits. A small processing unit effectively and rapidly simulates the cellular automata. Such a processor would occupy a small fraction of the area on a typical integrated circuit. Using VLSI greatly reduces the cost of electronic components.

3 The Fluid Dynamics Machine Architecture

The fluid dynamics machine consists of three major parts: the processing elements, the intercell communications system, and the input-output system. The machine contains 9216 processing elements. Each processing element calculates the new state for 1024 cellular automata cells. The processing elements are arranged logically into a square of 96 by 96 processors. The inter-processing element communication system moves particle data between adjacent processing elements when required. The input-output processor handles data that falls off the edge of the array and deals with the outside world, which in this case is a host engineering workstation. Communication between the processing elements is central to the systolic nature of the FDM architecture and is the part that can be the most confusing, so we will start with it.

3.1 Inter-processing Element Communication System

The inter-processing communications system allows several types of data to move from one processing element to another. The needs of particle data communication determined the number and direction of these data paths, but the paths are used to move many types of data. In addition to moving cellular automata particle data, the inter-processing element communication system must also provide paths for loading initial data into the machine, changing the collision rules and getting the results out of the machine. Theses paths are kept as short as possible by connecting only adjacent processing elements. By using this systolic architecture, little delay is associated with these data paths.

3.1.1 Particle Data Path Description

The communications go over unidirectional bit serial lines between processing elements that are connected to form a seamless hexagonal grid of processing elements. The best shape for a processing elements is a rectangle, and the best shape for the array is rectangular with aligned rows and columns. These implementation preferences require an interesting interconnection scheme.

To design these connections, all cases of inter-processing element communication must be examined. If the dark, solid lines in Figure 1 represent the boundary of a processing element, the required inter-processing element data connections may be seen. Consider the case of data moving after a collision out of a cell in the Northwest direction. For the top left cell in the processing element, communication with the processing element above and to the left is required. For the next cell to the right, communication with the processing element above is required. For the leftmost cell in the second row, communications with the processing element to the left is required. Figure 3 shows all of the required data paths of inter-processing element communication.

Each processor has four lines at the top and bottom and six lines on the left and right connecting to adjacent processing elements. Each processing element can send to six paths (two of the paths split to go to two different processing elements) and receive data from 10 different paths. Each output path corresponds to a direction of motion in the cellular automata model. Some of the input path directions are duplicated because data for a given direction of motion may come from as many as three different processors, depending on the location of the cell being processed. Figure 4 shows the how the processing elements fit together to form communication paths connecting nine processing elements.

3.1.2 Other Data Path Formation

The primary purpose for these paths is to move particle data during calculation, but this task occupies only a fraction of the available time. This particle data must flow between processing elements during the calculation of cells on the boundary of the processing element. There are 124 cells on the boundary, so cellular automata particle data must flow between processing elements 124 out of every 1024 computation cycles, 12% of the time.

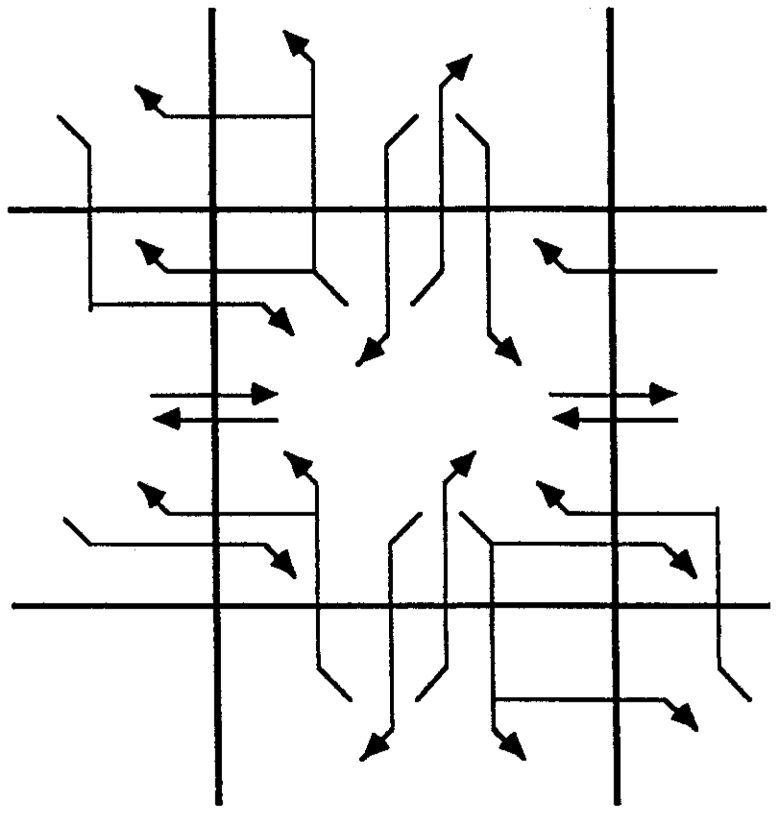


Figure 3: Inter-Processing Element Communication Paths

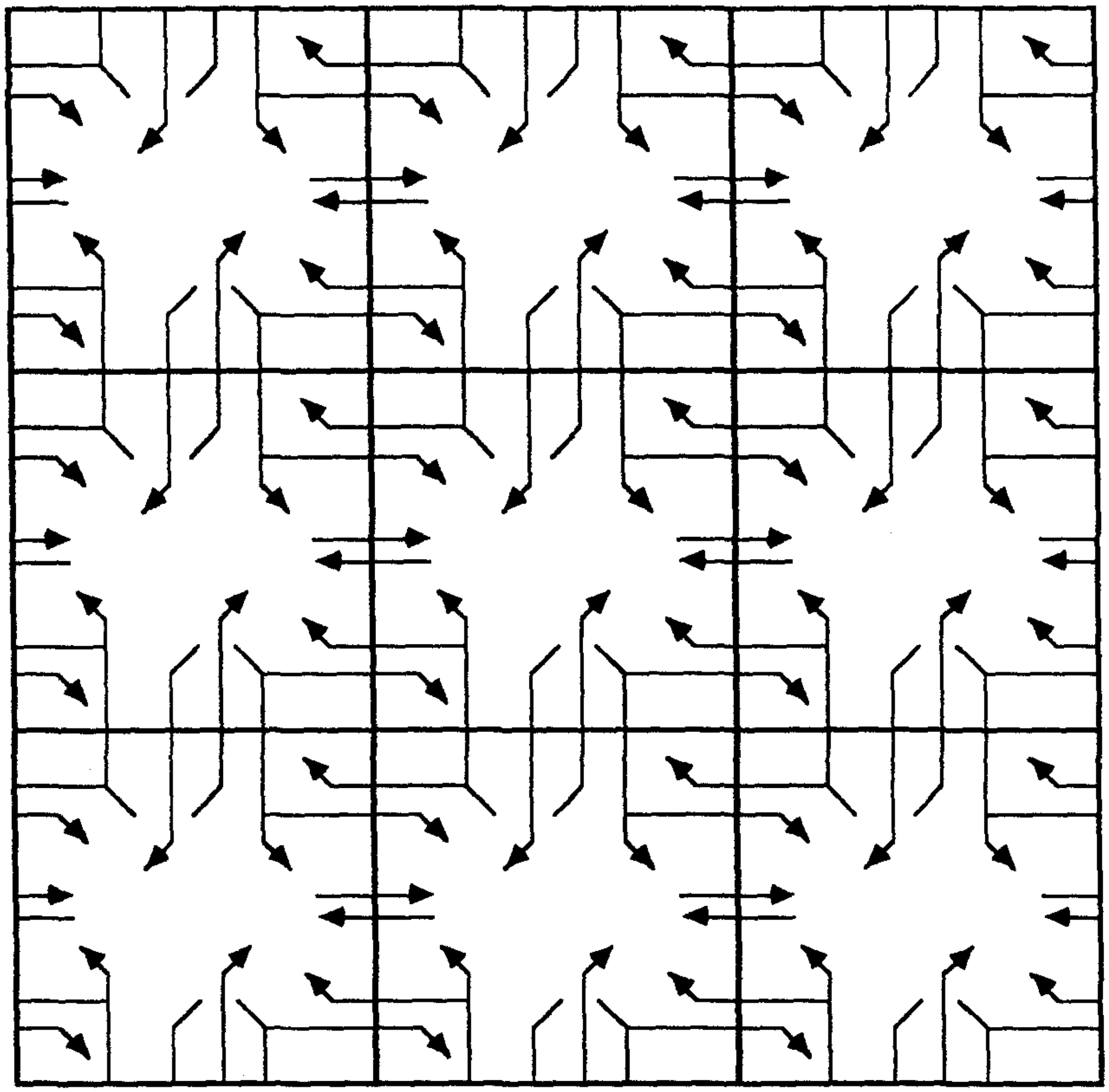


Figure 4: Nine Inter-Processing Element Communication Paths

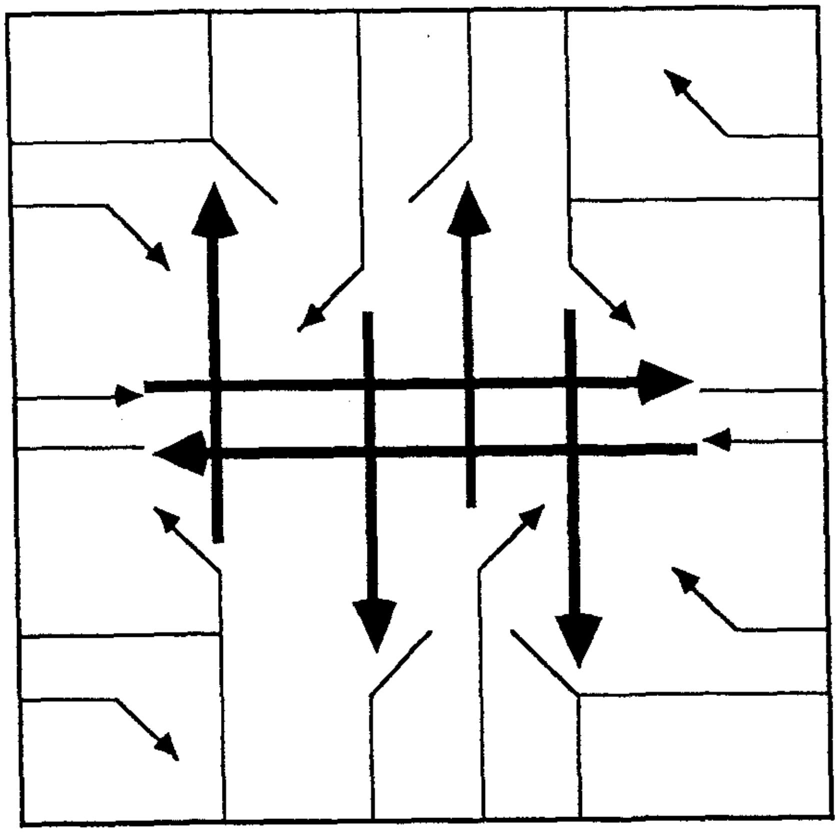


Figure 5: Data Path Interconnection to Form New Paths

By modifying the interconnection within the processing element, longer and wider data paths may be created. For example, the system could be used to transfer data other than cellular automata state data in a Northerly direction by using the Northeast, Northwest, Southeast and Southwest data paths. Each processing element would logically connect the Southeast input path to the Northwest output path. The data to be moved would reside in a shift register until the data paths were not occupied with cellular automata state data, then one bit of other data is shifted out. Figure 5 shows the connections inside a processing element to form two North, two South, one East, and one West data paths. To simplify the control logic, all directions of state data are placed on the data paths during each computation cycle that cellular automata must move, and other types of data are moved only when state data movements are not required.

3.1.3 Momentum Average Data

Although the results of the calculation may be obtained by examining the state of the machine after each calculation, the amount of data produced by the array of processing elements is too large and requires too much post processing for a conventional architecture host to use really well. A much better solution is to conduct some calculations inside the machine and periodically produce a smaller quantity of summary data. The Fluid Dynamics Machine calculates momentum average data, a smaller and much more usable form of result. Momentum average data is two numbers from each processing element indicating the average direction and magnitude of fluid flow for all the cells calculated by that processing element. Since each processing elements may calculate for thousands of cells (or sites) the amount of data is much reduced. The amount of data is still very large. Every 1024 cycles, 9216 pairs of 12 and 13 bit numbers are produced. The processing element uses spare time on the inter-processing element communication paths to move momentum data to the input-output processing system. That system then makes momentum average data available to the host processor at an appropriate rate. The Momentum average data is moved North and South out the array using the North and South data paths formed from the data paths as described above. Since two Northerly and two Southerly data paths are available, the 25 bits of momentum average data are broken into three 6 bit and one 7 bit packets. The 7 bit packets require 672 out of the available $1024 - 124 = 900$ cycles to move through 96 processing elements to get completely out of the array.

3.1.4 Loading Rule Data

The user must have a way to change the cellular automata collision rules stored in each processing element. The same North and South data path arrangement once again transfers this data.

3.1.5 Loading Particle Data

Before calculation begins, initial particle data must be placed in the array. After calculations are completed, the engineer may want to examine the final particle state. To get this data into and out of the array, it is shifted over this same North and South data paths.

3.1.6 Controlling the FDM

Certain control information must be sent to the processing elements and must arrive at all parts of the machine at one time. The delay introduced by this information slowly progressing from one processing element to the next through the normal data paths is unacceptable for these signals. Clock signals and global mode control signals (CALCULATE, RESET, LOAD STATE, and LOAD RULES) must be globally distributed. The global mode control signals may be encoded on two conductors, but the each phase of the two phase clock will require separate conductors. Also power and ground must be globally distributed.

3.2 Processing Element Description

The processing element calculates and stores the state of the simulated fluid. The machine consists of 9216 identical processing elements. This section will first describe the architecture of the processing elements and the use of pipelining to improve performance. Then it will describe the parts of the processing elements: the controller, the state memory, and the rule memory.

3.2.1 The Processing Element Architecture

This section will present how the Fluid Dynamics Machine implements the Frisch algorithm. FDM simulates the motion of the fluid using a set of collision rules defined by Frisch. Particles in the cellular automata are represented as bits in a particle state memory, and motion is simulated by moving those bits between memory locations. Particle data moves from an old state memory to a new state memory. Each processing element calculates the motion of particles in one cell each computing cycle. In addition to calculating fluid motion, the processing element receives and passes along rule, state or control data.

3.2.1.1 Simulating Motion of the Fluid The purpose of this computing machine is to simulate a fluid's motion using Frisch's algorithm in which particles represent microscopic portions of the fluid. Particles are located in cells and may be stationary or moving in one of six possible directions. The existence of all of the possible particles that may be in one cell is represented in one word of the new and old state memories. The state words consists of 9 bits: one bit to indicate a the presence of a stationary particle in the cell, six bits to indicate particles moving into the cell from each of the possible inter-cell directions, and two bits to indicate the applicable collision rule that applies to the cell. The motion of these bits representing particles inside a cell is accomplished by table lookup. An address is formed by adding the data in a state word to the single bit output of a random number generator to provide for the possibility of different collision outcomes required by Frisch's algorithm. That address is used to obtain a word from the rule table that contains 7 bits representing the existence of particles exiting the cell in the six possible directions and the existence of a stationary particle left in the cell. Since by convention the state memories contain information on particles entering a cell, new cell addresses must be calculated for particles exiting the cell. This address recalculation is easily implemented by the processing element. This method of simulating fluid dynamics using Frisch's algorithm is called table lookup.

Although other methods could be used to calculate new state, a lookup table permits total flexibility in rule construction at the expense of memory area. The machine calculates the expected outcome using the existing particles, and does not depend on any special relationships to calculate the results. The addition of randomly selected rules permits performance of the rules defined by Frisch and many similar rule definitions. Nonrandom rules are easily specified by providing the same rule entry for each of the possible random choices. This flexibility ensures the usefulness of the Fluid Dynamics Machine in case further research indicates that a small modification to the Frisch algorithm is beneficial.

By providing four sets of rules, the FDM can simulate multiple types of cellular automata collisions. These different rule sets may correspond to different media like free space, a rough wall or a shiny wall. By assigning different rules to some of the cells, the user may configure the problem with any shape of material desired.

3.2.1.1.1 Implementing the Rules The rule memory consists of 1024 words of 7 bits each, accounting for 128 particle combinations (2^7) times 4 media rules times 2 different random behaviors. In many cases, only free space and obstructed media would be required. The obstructed medium (which returns each particle along the path from which it came) can be used to construct a rough surface. To simulate a smooth wall, rules can be specified to reflect particles out along a different path. The rules must be encoded on the host machine and transferred into Fluid Dynamics Machine's rule memory before calculation begins. During calculation, each processing element will use the address generated from the state memory and random number generator to access the rule memory. The output of the rule memory is the new state.

3.2.1.1.2 Movement by Address Calculation After table lookup in the rule memory produces the new state for a cell, this new state must be distributed to several cells. FDM calculates the motion of the particles by writing the representative bits to the correct cell. Except for the stationary particle, the particles will move to adjacent cells, therefore the new particle data must be written to an adjacent cell's memory. The determination of the correct cell is made easier by intelligent arrangement of cells in the state memory. Figure 6 shows an arrangement of 16 cells, and the same techniques apply with the 1024 cells in one processing element. The memory is organized with an equal number of rows and columns, and each row is shifted one half position to the left of the row above. The address of any cell is simply the row number concatenated with the column number. To determine the address of the cell to the Southeast, simply add 1 to the column number and 1 to the row number of the current cell. For each calculation, the current row and column, the next and previous row and the next and previous column are required. These six quantities combine two at a time to form the required six new addresses. Because this arrangement allows the memory to be physically implemented as a square even though the logical arrangement is hexagonal, the state memory may use standard rectangular VLSI layout. All of the required operations described above are simple, so they can be rapidly executed on integrated circuits. This technique of movement by table lookup and address calculation uses standard VLSI components and rapidly calculates each new cell state.

The cells on the sides of the processing element add another complexity to addressing: some of the particle data must move to the adjacent processing element. Fortunately, the correct memory address in the adjacent processing element is the same as the address already calculated. When data must move between processing elements, the processors place all data on the outbound communication paths. Then, based on the current cell address, the processing elements determine whether to take local data or data from one of the adjacent processing elements. The processing element makes this calculation independently for each particle (corresponding to a direction of motion).

3.2.1.2 State Cell Memory Operations The Fluid Dynamics Machine must efficiently store the results of each calculation in the state memory. To improve performance, completely separate memories are used for the new state and the old state. As soon as the new state data is available, it may be written to the new state memory. At the same time, more old state can be read from the other section of the memory. Each FDM processing element has two state memories that are alternately assigned as old state and new state. After calculating all

```
             0\ /    1\ /    2\ /    3\ /
             --*-------*-------*-------*--
              / \0000 / \0001 / \0010 / \0011
             /   \   /   \   /   \   /
            /     \ /     \ /     \ /
         4\ /    5\ /    6\ /    7\ /
         --*-------*-------*-------*--
          / \0100 / \0101 / \0110 / \0111
         /   \   /   \   /   \   /
        /     \ /     \ /     \ /
     8\ /    9\ /   10\ /   11\ /
     --*-------*-------*-------*--
      / \1000 / \1001 / \1010 / \1011
     /   \   /   \   /   \   /
    /     \ /     \ /     \ /
12\ /   13\ /   14\ /   15\ /
 --*-------*-------*-------*--
  / \1100 / \1101 / \1110 / \1111
```

Figure 6: Logical Node Layout and Numbering

3.2.2 Performance Improvement through Pipelining

Pipelining, the technique of dividing a computation up into several stages, with output of each stage providing data to the next, is commonly used to improve the throughput of computer systems. Because each stage is working on a different problem, the pipeline will produce a result in the time it takes for only one stage of the work to be done. Since each stage of the pipeline has only a small amount of computation, it may run very fast. As a result the cycle time may be very short and the throughput of the machine will be high. Pipelines may be inefficient if they must be emptied and refilled frequently, but the pipelining scheme used in FDM does not suffer from this disadvantage. The pipeline requires only 4 cycles to fill, and only requires refilling when calculations are stopped. Typical expected applications would run many millions of cycles before stopping, so pipeline fill overhead would be on the order of 0.0001%.

The FDM uses five pipeline stages. In the first stage, a new address is applied to the old state memory. In the second stage, the old cell state is retrieved from the memory. In the next stage, the rule memory is consulted. In the fourth stage, the new state data from the rule memory is transmitted to its destination (which may be in another processing element) and the new state memory addresses are calculated. In the fifth stage, the new state data is stored in the memory.

Pipelining is made easier because separate memory is used for new state and old state. By carefully selecting the order of cell calculation, the pipeline has many cycles to update a memory cell before it will be accessed again. This allows ample time for data to reach its destination (even traveling between circuit boards) and prevents any pipeline interlocks while data is made valid. As a result, the pipeline remains filled and busy from initial pipeline fill when the host starts the calculations until the host stops the calculations.

3.2.3 The Controller Section

The controller located in each processing element tracks and controls all of its activities. To do these functions the controller contains a finite state machine and a considerable amount of random logic. It contains the counter to remember the current cell being processed. It contains the logic for data routing and address calculation, and it also contains a random number generator. Momentum average data is calculated in the controller and stored for shifting out of the array. The finite state machine selects the processing element mode of operation from the possibilities as directed by the global mode control signals of RESET, CALCULATE, LOAD RULE, and LOAD STATE.

3.2.3.1 The Controller Counter The controller contains a 10 bit counter that is maintained by a finite state machine, and selects the cell to be calculated next. The counter's contents are applied as an address for the old state memory and saved for later calculation of the new state memory address. The controller increments the counter each cycle during normal operation, and uses a method of incrementing that causes memory representing cells at the boundary to be accessed first, and cells internal to the processing element to be address last. One easy implementation of this strange incrementing is to invert the high order bit of the row and column address. This order of processing cells guarantees that enough cycles will pass for the old state memory to be updated before it is needed to calculate a new cell. The controller also sets the counter to zero when the global control signal RESET is active. Because the RESET signal overrides incrementing the counter, the RESET also functions as a halt.

3.2.3.2 Data Path selection The processing element also determines the use of the interprocessor communication paths. During calculation, two uses are possible. If the current cell is on a boundary, the new state data is placed on the these paths. Otherwise, momentum average data is shifted North and South. During rule load, the rules are shifted through the rule memory using the North and South data paths. During state loading, the old state is shifted out and new state is shifted in. Because rule and state data shift through the array, the data in the array at the beginning of the shift may be retrieved by the host workstation. Because of the large amount of data involved, this process is very slow, but this is not serious because it will only be done rarely.

3.2.3.3 Address Calculation The controller contains two incrementers and two decrementers to calculate the previous and next row and column addresses. The stored cell address and the output of these units are combined to form the seven storage addresses for the new particle data.

3.2.3.4 Random Generation

The processing element controller contains a linear feedback shift register. The pseudo random one bit output from this register is used to select potentially different collision rules. When a collision has a deterministic outcome, the two rules selected by this random bit will be the same.

3.2.3.5 Momentum Average Data Calculation

The controller accumulates momentum average data in two registers. The North register is incremented once for each Northwest or Northeast particle and decremented once for each Southwest or Southeast particle. The West register incremented once for each Northwest or Southwest particle and decremented once for each Northeast or Southeast particle. The West register is incremented by two for each West particle and decremented by two for each East particle. The East and West particles have twice the effect on the West register compared to the other directions because of geometry, since the other directions would have to be multiplied by the sin(30 deg) which is 0.5.

When the data has been received for all cells, the momentum data is transferred to four shift registers. The North data is split into two 6 bit registers. The West data is placed in a 6 bit and a 7 bit register. These shift registers connect to the inter-processing element communication paths and are used to shift the momentum average data out of the processing element array. At the same time, the North and West registers are reset to zero.

3.2.4 The Cell State Memory

The Fluid Dynamics Machine uses a specialized memory system. The cellular automata state memory of each processing element has two modules containing 1024 words each 9 bits long. Each module has separate write addressing for each bit, and the module handles either 7 bit or 9 bit writes. The 7 bit writes are used during calculation when the two bits representing the rule to be used for that cell are to remain unchanged. During state load, the rule associated with each cell may change, so a 9 bit write is also provided. The two memory modules are connected to the controller by a crossbar switch. This switch determines the old state module and the new state module. Because the switch provides enough data paths and the memories are independent, read and write operations may take place at the same time.

The four shift registers mentioned above in the momentum average section are used to hold incoming and outgoing state data during load. Data is shifted in for three cycles with the cell address counter held constant. Data is then simultaneously written into the one state memory are read from the other state memory.

3.2.5 The Rule Memory

The rule memory contains the collision rules (or next state rules) for the Fluid Dynamics Machine. The memory is big enough to provide a randomness selection and four different rule sets. The memory is a conventional read-write memory without any remarkable features.

Once again, the four shift registers mentioned above in the momentum average section are used to hold incoming and outgoing data during load. Data is shifted in for two cycles with the cell address counter held constant. Since the rule memory cannot be read and written in the same cycle, old rule data is read on one cycle and new data is written on the next cycle. Since the four shift registers are big enough to hold this data, only time (one cycle) is required to handle this problem. This one additional cycle added to the thousands of cycles required to load the rules is insignificant.

3.3 The Input-Output Controller Description

The input output controller is a special purpose cpu to handle the communications requirements of the Fluid Dynamics Machine. It has its own memory, registers and data busses. It communicates with the array of processing elements and with the host engineering work station.

Since the input-output controller is a standard design computer, we will present it by covering the features, the address spaces, the formats of data and instructions and the operations that it can do.

3.3.1 Unusual Features

Although the input-output processor operates like a general purpose computer, it has several unusual features. It has the ability to stop and restart the processing elements in the Fluid Dynamics Machine array. It transfers data over a bus that varies the function of each conductor to match the type of data transfer. To communicate with the host work station, the input-output processor has several registers that may also be accessed from the host workstation's data bus.

3.3.1.1 Clock Control by the Input-Output Processor One common problem with systolic arrays is the very high peak production rate of data at the boundary of the array. To control the flow of data at the boundary, the input-output processor can start and stop the clock supplied to the processing element array. The input-output processor runs at the same clock speed as the array, and it interprets a new instruction each clock cycle. As it decodes an instruction, it determines whether the clock signals will go to the processing element array during this input-output processor clock cycle. Because control of the array clock is encoded in the instruction word, the clock is under the control of the input-output processor programmer.

3.3.1.2 A Multifunction Data Bus The input-output processor uses a specially designed bus for data transfer. Many features are included on this bus because the major purpose of the input-output processor is to move data around. The multipurpose data bus uses a different number of conductors for address specification and data depending on the mode of operation. It has modes of operation that read and write from memory simultaneously. These features speed up and reduce the amount of hardware required for different types of data transfer.

3.3.1.3 Host Access

A set of registers may be accessed from either the host or the input-output processor. These registers convey control information to the Fluid Dynamics Machine, data to the processing element array and data result data to the host.

3.3.2 Address Spaces

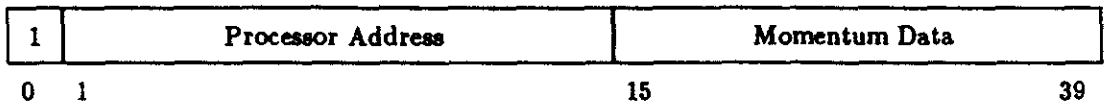
The input-output processor uses several spaces. One address space contains the 32 bit long boundary registers. The same register addresses are used for read and write. A read gets data from the array of processing elements, while a write sends data into the array. Thus a read will not always produce the same data just written to the same register address. Three of these registers are wide enough to provide one bit of input and output data to each processing element. Each side of a processing element requires two bits, however, and there must be registers on each side of the array. Thus Fluid Dynamics Machine has 24 of these registers. Another address space identifies the momentum average memories. These are 25 bits long and there are 9216 of them (one for each processing element). The last address space contains the 32 bit long VME bus interface registers that specify addresses, data and control information. Eight of these are available, but currently only 5 are in use.

3.3.3 Description of Multipurpose Data Bus

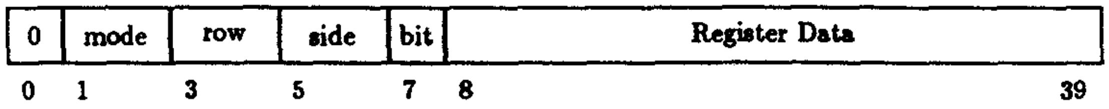
The multipurpose data bus consists of 40 conductors and operates in register transfer mode or momentum data mode. The address-data partitioning of these conductors depends on the value of the first address bit. In register transfer mode, 8 conductors specify the address and 32 lines transfer data. In the register transfer mode, the bus can perform conventional operations like fetch from memory an place in accumulator and store from accumulator into memory. In this mode it can also perform two special operations that simultaneously read data from one side of the processing element array and write it to the other side of the array. In momentum data transfer mode, 15 bits specify the address and 25 bits of data move along the bus. Figure 7 shows the formats of the multipurpose data bus.

In momentum mode, the momentum data is mapped onto the locations 0 to 9215. Attempting to address any other momentum data is an error. The four possible operations identified in bits 1 and 2 in the register transfer mode are explained in the data handling operations section below. The register transfer mode is also used to access the VME bus registers. When the row bits (3 and 4) are zero, then bits 5, 6, and 7 identify VME bus registers. They are assign as follows:

- 001 Processing Element Array Command
- 010 Boundary Register Address (only low order 8 bits are valid)
- 011 Boundary Register Data (as seen by host)
- 100 Momentum Address (only low order 14 bits are valid)



Momentum Mode



Register Transfer Mode

Figure 7: Multipurpose Data Bus

- 101 Momentum data (two 16 bit numbers)

All other addresses in row 00 are invalid and cause an error if addressed. These 32 bit registers may be accessed by the host using memory addresses on the VME bus.

3.3.4 Working Storage

Working storage consists of the following registers:

- 1 32 bit data register (also referred to as the accumulator)
- 1 10 bit clock register
- 1 10 bit clock interrupt register
- 1 10 bit clock interrupt address
- 1 10 bit clock interrupt return program counter
- 1 10 bit program counter
- 2 1 bit mode indicators
- 1 40 bit multipurpose data bus.

Figure 8 shows the register model for the input-output processor. Several data paths are shown between the Multipurpose Data Bus and the Data Register because different alignments of the data transferred are used for different operations.

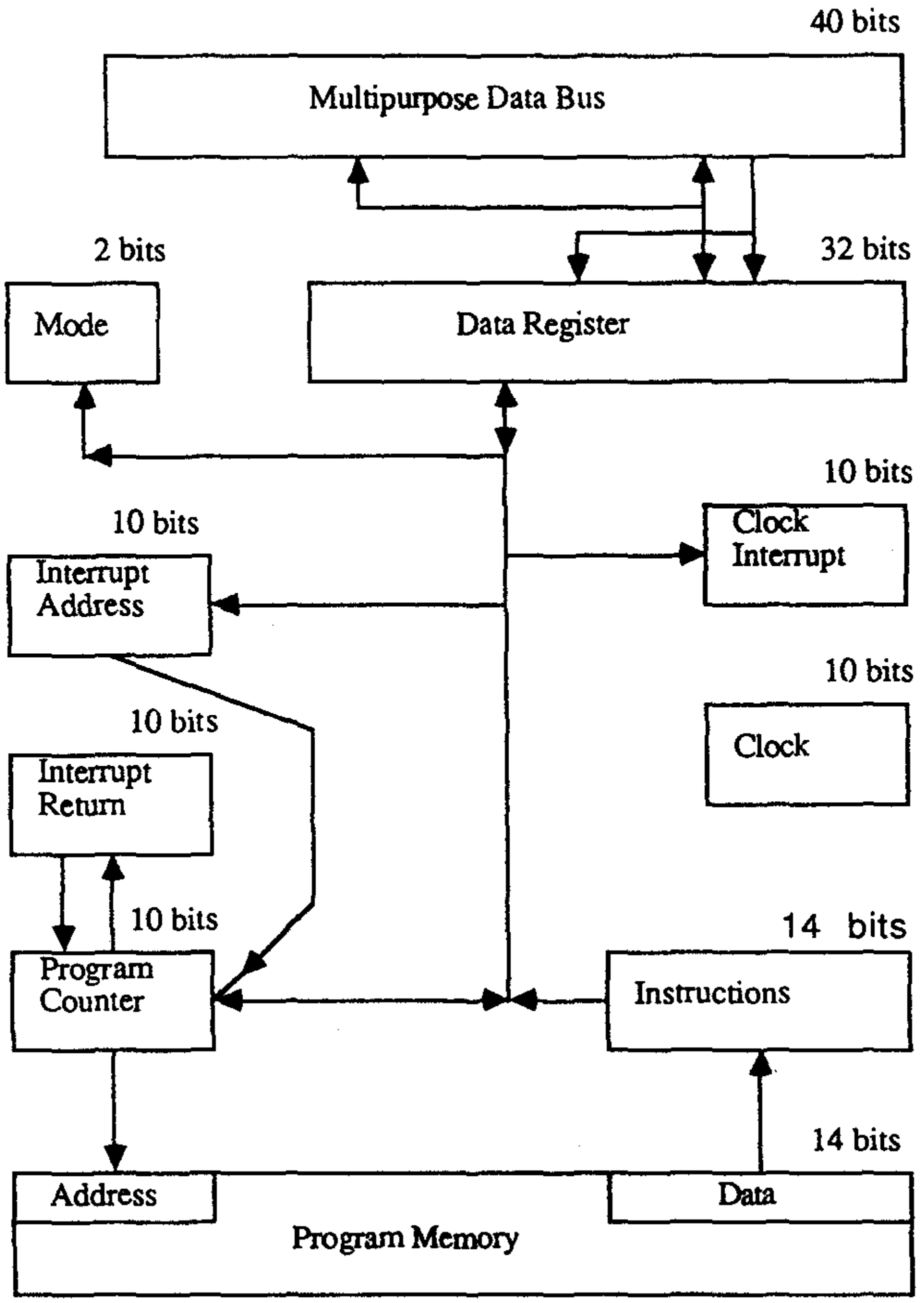


Figure 8: Working Storage Register Model

3.3.5 Formats of Data and Instructions

This section will describe the formats of data and instructions.

3.3.5.1 Data Formats The input-output processor uses four types of data:

1. internal momentum data
2. external momentum data
3. cellular automata data
4. immediate data.

Momentum data that comes out of the processing element array consists of a 13-bit two's complement number representing average West direction momentum and a 12-bit two's complement number for the North direction. Because the processing element forms these numbers by addition of particles along a hexagonal grid, the East-West component of the momentum average data is on a different scale from the North-South data. This results in the West direction requiring more bits for representation than the North direction. The factor is the sin(60 deg), which is $\sqrt{3}/2$. To correct this scale factor would require the each processing element to contain a multiplier. Such a multiplier would only be used once every 1024 processing element cycles, because the scaling would only be required after all momentum average data had been accumulated. Multipliers occupy too much area for a feature used so infrequently.

Momentum data is converted to two 16-bit two's complement numbers when it is presented to the host computer. The West component is placed in the 16 high order bits of the 32-bit VME bus word and the North component is place in the low order 16 bits.

Cellular automata data is 32 bits long. The input-output processor transfers the 32-bit data to and from the array boundary registers and the VME bus registers.

Immediate data, located in the instructions, may be 8 or 10 bits long.

3.3.5.2 Instruction Formats FDM has two types of instructions, move instructions and register instructions. The two instruction formats are shown in figure 9.

3.3.6 Operations

Only those operations required for the limited requirements of a simple input-output processor where included. Six operation codes are used; however the Move instruction has submodes which effectively give a variable length operation code and a total of about 14 operations.

clock	000	type	Address Data	
0	1	4	6	13

Move Instructions

clock	opcode	Address or Data	
0	1	4	13

Register Mode

Figure 9: Instruction Formats

3.3.6.1 Data Handling The move operation is the most basic and most used. It has four modes:

- "move immediate to data register" with 8 bits of immediate data.
- "move immediate to operation mode" with 2 bits of immediate data.
- "move momentum data" from memory to data register - address contain in data register.
- "register transfer" with several submodes (see below).

The "move immediate to operation mode" selects one of the operating modes for the entire Fluid Dynamics Machine. Possible modes are RESET, CALCULATE, LOAD STATE and LOAD RULE. Loading the operation mode sends the appropriate global control signal to the processing element array.

The register transfer mode further breaks down to the four multipurpose data bus submodes of operation. These submodes are indicated by the high order 2 bits in the address. These two bits and the following bits are place in bits 1 through 7 of the multipurpose bus. The submodes are:

- 00 read boundary register to data register.
- 01 read boundary register and write to opposite side boundary register.
- 10 read boundary register to data register and write to opposite side boundary register.
- 11 write boundary register from data register.

3.3.6.2 Sequencing and Decisions Two instructions are used for sequencing and decision. The "skip on mask" instruction will cause the following instruction to be skipped if the logical and of the immediate data and the low order 10 bits of the data register is not zero. The "unconditional branch" transfers control to the address specified in the instruction. A "skip on mask" followed by an "unconditional branch" make a conditional branch. This technique was used to avoid creating condition codes. Without using condition codes, a conditional branch would not fit in the small instruction word.

3.3.6.3 Supervisory Provisions FDM has a clock interrupt. A counter is driven by the same clock sent to the array of processing elements, so it contains the same value as the counters in all the processing elements. When this counter contains the same value as in the clock interrupt register, the value in the clock interrupt address register is loaded into the program counter. The old contents of the program counter are saved in the clock interrupt return program counter. To support these features, the input-output processor uses the following instructions:

- "set next clock interrupt time" from immediate data.
- "increment next clock interrupt time" by immediate data.
- "set next clock interrupt location" from immediate data.
- "return from interrupt" - reload the program counter with the previous value

The clock signals provided to the processing element array are directly under input-output processor program control. A bit in each instruction specifies whether the clock is to be active during this instruction cycle. This is indicated to the assembler by specifying (clock) at the beginning of each instruction requiring a clock signal to the processing element array. The main use of this facility is to stop the array long enough to move data from one side of the array to the other. The programmer is responsible to minize the time without clock signals to the processing element array to prevent loss of data in dynamic circuits.

The input-output processor provides for three possible errors:

1. invalid opcode.
2. invalid register address.
3. invalid momentum data address.

On occurrence of any of these errors, the program counter is stored in the data register (presumably to be transferred to the host) and the program jumps to location 0.

4 Wafer Scale Implementation Considerations

This machine is made up of an array of 9216 identical processors that each take up relatively little silicon area. This situation is well suited to wafer scale implementation. Putting the array on wafer sized pieces of silicon simultaneously reduces cost and improves performance. Cost will be reduced because fewer packages and less interpackage communication is required. Although one wafer costs significantly more than standard size intergrated circuit, it costs less than the cost of the large number of integrated circuits it replaces. Wafer scale implementation also saves greatly on the cost of circuit board and wiring. The machine's performance will be better because the inter-processing element communication in most cases will be to processing elements adjacent on the same piece of silicon. Short, low capacitance wires connect adjacent processing elements, so there is no need for large drivers to support inter-processing element communication and performance is faster. Also, since many large drivers are not required, the chips use silicon more efficiently and have lower power consumption and cooling problems.

The Fluid Dynamics Machine has many desirable properties for wafer scale implementation, but many parameters of the machine must be chosen to optimize wafer scale implementation. For example, if each processing element is so big that it occupies a complete wafer, the machine will be far too expensive. Only a tiny portion of wafers do not have at least one flaw, so it would be uneconomical to make enough to find one without flaws. Therefore, the parameters of processing element function (as represented by area), fault density on the wafer and communication requirements must be balanced.

4.1 Area - The Parameter Under Designer Control

Area is an important parameter for wafer scale design because function and probability of faults depend on area. The architect can specify the area of the wafer and more importantly, he can control the area of the processing element. Proper selection of the size of the processing element improves the advantage of wafer scale implementation because it effect on yield.

4.1.1 Size of the Wafer

The size of the wafer will determine the number of processing element on each wafer. The available fabrication process may also depend on the size of the wafer selected.

4.1.2 Size of the Processing Element

The capability of the processing element determines its area. The processing element must have enough capability to function, but the size of the state memory may be controlled by the implementor. In a trial design with only 16 cells, the state memory occupied 40 percent of the area. Better memory design will require less area, but state memory will be a major factor in the total area of the processing element.

4.2 Fault Distribution

A fault is an imperfection on a wafer. A fault may result in a broken wire, a short between two wires or active components that are inoperative. Wafers made by commercial fabrication processes typically have 2 to 5 faults per square centimeter. As a result, average wafers have hundreds of faults, and the production of a fault free wafer is almost statistically impossible.

A simplified model will be used for faults that may occur in fabrication. Our knowledge of faults is limited because fault data is closely guarded proprietary information. The figure of 2 to 5 faults per square centimeter is derived from private conversations with individuals, but more concrete references are not available. Additionally, we want to model against many different production lines at varying levels of maturity. Given these goals, a simple model will be the most useful in much the same way that lambda type design rules serve MOSIS users so well. I will assume that the faults are evenly distributed over the wafer. Recent research indicates that the fault distribution is uneven, but the fault recovery technique to be used works well with multiple faults in close proximity. As a result, the even distribution model will give conservative results. I will also assume that the size of each fault is relatively small, 50 microns, so the chance of two processing elements being effected by the same fault is small.

4.2.1 Recovery Potential of a Wafer

To recover from faults, the Hedlund algorithm[4] will be used. This algorithm will provide satisfactory routing around defective cells as long as 60 percent of the cells are available. As a consequence, the wafers must be designed to provide at least a 60 percent yield.

4.3 Input-Output Requirements

Kung[5] noted that communications requirements are effected by processing element memory size. The Fluid Dynamics Machine follows this trend. Inter-processing element communication is required for each processing element on the boundary. As such inter-processing element communication during state calculation is proportional to the square root of the memory size. Since the processing element can calculate a new cell state in one cycle, we have significant communications overcapacity, however, the same data paths are to be used for momentum data shifting. The size of the momentum data is proportional to the log of the memory size since the maximum momentum occurs if all cells have particles going the same way and log (memory size) bits will store that number.

4.4 IO and Yield calculations

The Fluid Dynamics Machine meets the requirements for communications and yield when configured with 9216 (96 by 96) processing elements and 1024 cells in each processing element.

Communications requirements are met with 1024 cells in each processing element. In 1024 cycles, 124 cycles are used for state data movement (the number of cells on the boundary of a processing element) and 672 are used for momentum average data movement (7 bits across 96 processing elements). This leaves 228 extra cycles for use by the input-output processor.

Assuming that a 1.2 micron process will be used, a fault density of 5 faults per square centimeter is appropriate. A five inch wafer (a medium size in 1987) provides a 9cm by 9cm square, 150,000 lambda by 150,000 lambda. A processing element containing the rule set described above and 1024 state memory words would occupy a rectangle 3000 by 4000 lambda. A wafer then contains 1875 processing elements. The same wafer contains 405 faults (5 faults per square cm times 81 square cm). The chance of a processing element being good (the same as the percentage of all processing element's that are good) is (the number of processing elements-1)/(the number of processing elements) raised to the power of the number of faults on the wafer.

In the case of the Fluid Dynamics Machine, the yield is $(1874)/(1875)^{405} = .8056889$. This yield will amply support restructuring around faulted processing elements. Allowing 20 percent loss for restructuring, an average of 60 percent of the processing elements will remain active after restructuring, so each wafer will provide 1125 processing elements. Nine wafers give 10125 processing elements, more than enough for a Fluid Dynamics Machine.

5 Summary

Clearly, the Fluid Dynamics Machine rapidly and effectively solves the fluid dynamics problem. It will calculate 10 million cellular automata propagations in 1024 cycles. This machine should operate with a 25 megahertz clock, so the 10 million calculations could be performed in 41 microseconds. The Fluid Dynamics Machine would easily provide data faster than an engineering workstation could display it. The performance is high enough that an engineer could get data in so short a time that he would consider it instantaneous. The cost to manufacture would also be low. A single circuit board containing the nine wafers and support circuitry including the input-output processor could be manufactured for about 5 thousand dollars, a price within reach of thousands of interested users.

6 Acknowledgements

The algorithm adaptation and demonstration simulations were done by Kye Hedlund. The architecture was designed by Kye Hedlund, Mark Davis and Mike Kotliar. The processing element was designed by Mark Davis and Mike Kotliar. The input output processor was designed by Mark Davis.

References

[1] Frisch, U., Hasslacher, B., and Pomeau, Y. "A Lattice Gas Automaton for the Navier Stokes Equations," Los Alamos Nat. Lab., LA-UR-85-3503.

[2] Hedlund, K.S. AND Snyder, L. "Systolic Architecture - A Wafer Scale Approach," *Proceedings IEEE International Conference on Computer Design*, (Oct. 1984), 604-610.

[3] d'Humieres, D. AND Lallemand, P. AND Shimomura, T., "Lattice Gas Cellular Automata, A New Experimental Tool for Hydrodynamics" Los Alamos Nat. Lab., LA-UR-85-4051.

[4] Hedlund, K. "The Design of a Prototype WASP Machine" *Wafer Scale Integration*, Saucier, G. and Trilhe, J. (Editors) (1986), 89-97.

[5] Kung, H. T. "Memory requirements for Balanced Computer Architectures" *IEEE Conference Proceedings of the 13 Annual International Symposium on Computer Architecture*, **14-2**, 12 (June, 1986), 49-54.

[6] Mead, C. and Conway, L. *Introduction to VLSI Systems*. Addison Wesley, Reading, Massachusetts, 1980.

[7] Seitz, C. L. "Concurrent VLSI Architectures" *IEEE Transactions on Computers*, **C-33**, 12 (Dec. 1984), 1247-1265.