ADINA Computer I : I. Architecture and Theoretical estimates

AUTHOR(S):

NOGI, Tatsuo; KUBO, Masatoshi

ADINA Computer I†

I. Architecture and Theoretical estimates

by

Tatsuo NOGI* and Masatoshi KUBO**

(Received June 30, 1980)

Abstract

We propose a new form of architecture, a parallel computer, especially for the simulation of physical phenomena. The computer is a hierarchy composed of a host computer and N slave processors, with their own memories. It has a two dimensional array of $N \times N$ buffer memories being provided for transferring data among the slave processors. We give some theoretical estimates, which show a very high efficiency.

§ 1 Introduction

Until now, many people proposed multi-processing computers in agreement with an idea that each processor deals with a point or a block of some neighbouring points on a lattice region for simulation. Among them two dimensional array computers of the SIMD type (for example, ILLIAC-IV) having processors which execute microprograms with the same number of steps synchronously in control of a host computer, have demonstrated their ability for fairly limited methods of computation. However, in general, they do not show a high efficiency because their configurations are inconvenient for transfering data between separate processors, and the control is complex.

Some other computers have common memories and common buses for a direct transfer of data between any two processors. In these computers, problems of bus competition and memory competition are solved in the manner of time-shared buses, crossbar switches and multiport memories. However, such methods have defects of low efficiency, abundant hardware and expensive memories, respectively.

Moreover, there are hybrid computers of the SIMD and MIMD types. In SMS 201, for example, each processor has an exclusive part in its private memory instead of a

† Alternating Direction Immediate Nexus Array Computer I

* Department of Applied Mathematics and Physics

** Department of Information Science

corresponding part of common memory, renews contents of memories in the way of broadcasting and executes asynchronously. These kinds of computers are still limited for usage.

In our computer ADINA, each processor of one (or two) dimensional array deals with two (or three) lines of lattice points. The assignment is based on the judgement that it is efficient for solving implicit schemes, matrix calculation and the FFT algorithm, etc.. On the other hand, the ADINA takes the form of an asynchronous MIMD type in order to cut down the amount of hardware as much as possible. Also, a two dimensional array of buffer memories (or a set of some arrays) is provided so that data may be directly transfered between any two processors.

§ 2 Architecture

We state the fundamental architecture of the ADINA computer only in the case of a one dimensional array structure of processors. We shall state our computer of a two dimensional array structure (which we call ADINA-II) in the next paper.

This computer system is a hierarchy composed of a host computer and N slave processors with their own memories. It has a two dimensional array of $N \times N$ processors being provided for transfering data among the slave processors during parallel processing. As buffer memories, either the RAM (Random Access Memory) or the FIFO (First-In-First-Out Memory) can be used. Usually, it is more efficient to use the RAM, but the address buses from the processors to the RAM increase the quantity of hardware. Therefore, the RAM is not advisable for a case where many processors formed in a one dimensional array. On the other hand, the use of the FIFO may lessen the efficiency a little, but it decreases the quantity of hardware. If we use DMA channels for connecting the processors and buffer memories, the loss of efficiency is not so serious, and it becomes easier to act asynchronously. Also, the same FIFO memories are available for the transfer of data between the host processor and slave processors. In this paper, we consider only the case of using FIFO memories.

Now, we will explain the configuration of ADINA-I. In the following, the symbols MU, AU-j (j=0,1,$\cdots$,N—1) and FIFO-(i,j) (i,j=0,1,...,N—1) denote the host processor (the Master Unit), the slave processors (the Arithmetic Unit) and the buffer memories. The data bus from the MU is connected to all FIFO memories as seen in Fig. 1. Usually there is no necessity for its being connected to every FIFO. It is sufficient to be done only to the FIFO-(j,j) (j=1, 1, $\cdots$, N-1) on the diagonal of the array. However, we prefer connecting to every FIFO so that we may have a number of uniform boards on which some buffer memories being connected to the MU and some AU's are equipped. The AU-j is combined with the FIFO-(i, j)'s (i=0,1,...,N—1) on the j-th row of the array, and also with the FIFO-(j, k)'s (k=0, 1, ..., N—1) on the j-th column through

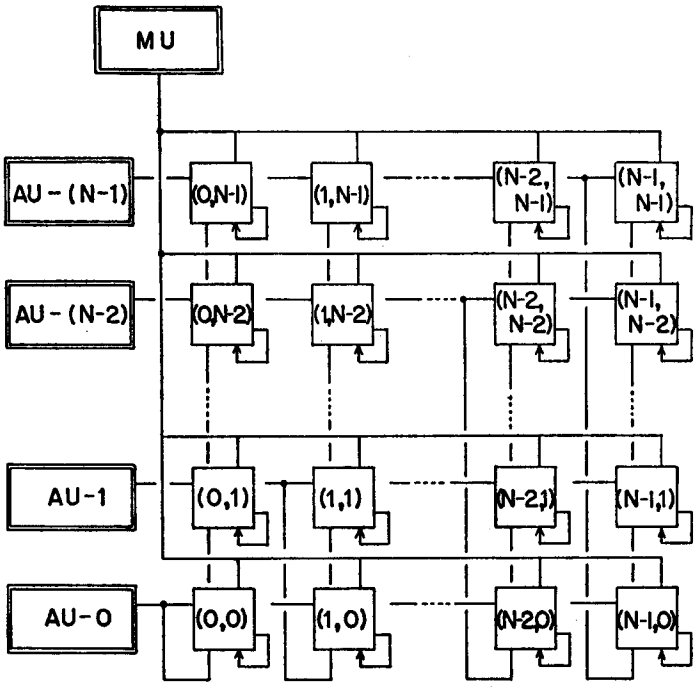

Fig. 1. Path-building of the ADINA Computer I.

data buses. Each FIFO buffer memory has its own loop bus combining its exit to its entrance. Fig. 2 shows the form of a bus connection to a FIFO-(i, j). It is seen that there are four ways from the MU, the FIFO's exit, the AU-i and the AU-j, and also the same number of ways to the MU, the FIFO's entrance, the AU-i and the AU-j. The loop bus is used so as not to lose the contained data in the FIFO when transfered to the MU. If it is not necessary, the loop bus may be cut.

In such path-building it is possible to transfer data directly among the MU and every AU, so that the ADINA computer becomes very useful for a wide range of problems. Moreover, it can exchange data mutually at any time between any two AU's. In fact, when the AU-i and AU-j must exchange data mutually, both actions that the AU-i sends to the FIFO-(i, j), and then receives from the FIFO-(j, i), and that the

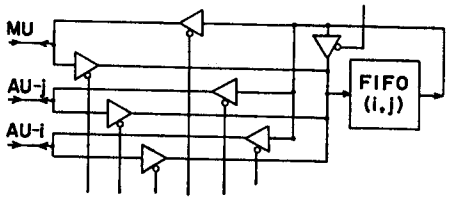

Fig. 2. Data buses to and from the FIFO-(i, j).

AU-j sends to the FIFO-(j, i), and then receives from the FIFO-(i, j) are taken parallel. It should be also mentioned that, since FIFO memories are used, an AU engaged in a data exchange need not catch its partner's state directly, but only catch a signal of the input ready, or the output ready, from the FIFO memory standing between them. Then, they can continue other computations until transfered data is needed. This is because the DMA controllers themselves play the role of receiving the signals.

As the ADINA computer is free from unified control and runs asynchronously, it needs only a small number of signal lines. On the other hand, though its path-building is of the crossbar type, the number of kinds of the data buses connected to a FIFO memory from the exterior is only 3 (or essentially 2 if it had other channels between the MU and each AU). Since the quantity of its hardware is not so much, we can therefore hope that the ADINA computer will open a way of increasing the number of processors.

§ 3 An example of trial maunfacture

For a trial manufacture of ADINA-I, we have selected the F8 microprocessor as each unit module among the inexpensive processors on the market. We consider a configuration as seen in Fig. 3. It consists of a host processor MU, 16 slave processors AU-j ($j=0,1,\cdots,F$) and a 16×16 two dimensional array of FIFO memories. The MU consists of a CPU (3850), a DMI (3852), two DMA's (3854), a PIO (3861), 4 K byte ROM's (2716) and 8 K byte RAM's (4027). One of the DMA's controls the sending and receiving data to and from the slave processors, while another controls those to and from an I/O processor. The PIO is provided for direct input and output by the MU itself, and for receiving interrupt requests from the AU's.

All the AU's have the same organization, and each of them has a CPU, a DMI, a DMA, 2 K byte ROM's and 4 K byte RAM's.

Each FIFO memory is composed of two chips of a 4×64 bit FIFO (2841). The signals of SHIFT-IN-REQUEST (SI REQ) and SHIFT-OUT-REQUEST (SO REQ), coming to the entrance and the exit of FIFO respectively, are accepted independently. On each side, when a processor is engaged to the FIFO, another processor is not accepted and must wait. The SI REQ and SO REQ are made from data latched on PORT 1 (P1) of each CPU, the ENABLE and DIRECTION from each DMA. When the MU is made to be engaged to the FIFO-(i, j), two digits of a hexadecimal number 'ij' are put on the P1. When the AU-j is made to be engaged to the FIFO-(i, j), one bit and a hexadecimal digit '$0i$' are put on the $(P1)_{4\sim0}$ (the bit 4$\sim$bit 0 of P1). When the AU-i is made to be engaged to the FIFO-(i, j), '$1i$' is put on the $(P1)_{4\sim0}$. These data must be latched before the DMA controller is made to start.

Between the MU and each AU, some signals are sent through their PORT 0's (P0's). When the MU is made to exchange data with the AU-j, the same hexadecimal

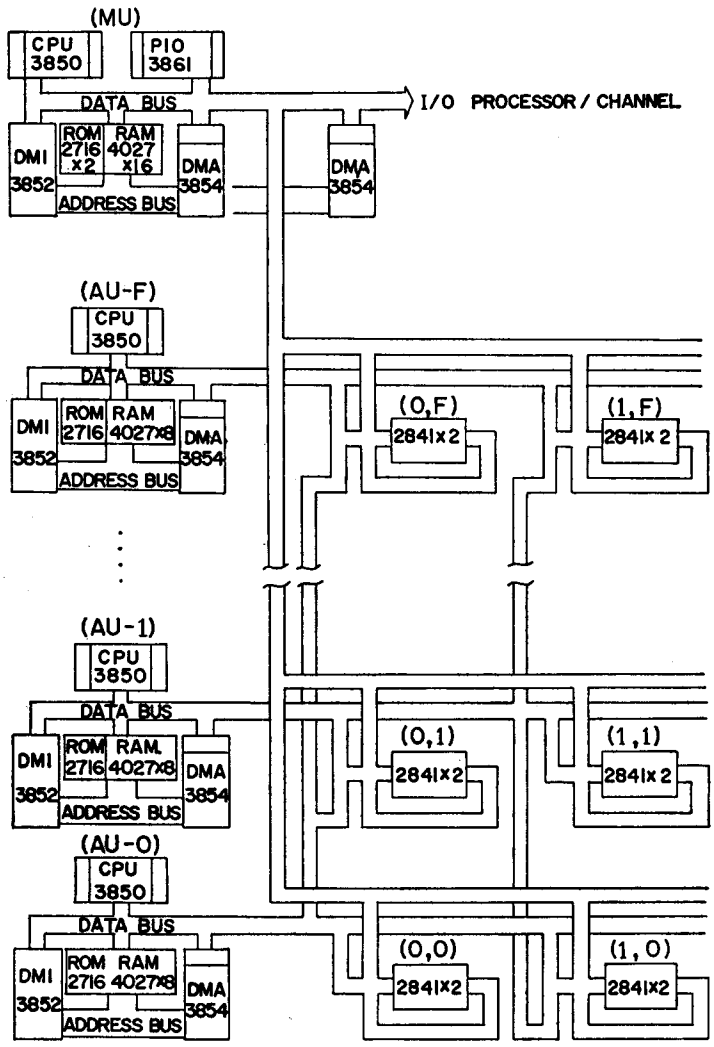

Fig. 3. Configuration of the trial manufacture.

number '*j*' must be latched on the $(P0)_{3\sim0}$ of the MU. In addition, the '1' of the $(P0)_6$ of MU means an instruction for the AU-*j* to start calculation, the '1' of the $(P0)_4$ means an instruction to return to the state of receiving other programs. The '1' of the $(P0)_5$ means broadcasting the same instruction by the $(P0)_6$ or $(P0)_4$ to every AU. On the other hand, the MU's receiving '1' on the $(P0)_7$ is taken as the end of a calculation shared by the AU-*j* when the $(P0)_{3\sim0}$ has '*j*'.

The AU-*j*'s putting 'l' on its $(P0)_7$ means the end of a calculation, and putting 'l' on its $(P0)_5$ means an interrupt request to the MU. On the other hand, the AU-*j*'s receiving on its $(P0)_6$ is taken as an instruction of starting and receiving 'l' on the $(P0)_4$

as an instuiction of stopping the calculation and returning to the state of receiving other programs from the MU.

The ROM of each AU contains some simple utility programs and subroutines of floating point operations.

Usually, it takes the following pattern to use the ADINA computer. The MU first gets a source program from the I/O processor, stores it in its RAM, next broadcasts a part of it to each AU, and then commands them to start. When some results appear in the FIFO's ,the MU receives them and sends them to the I/O processor.

Here we give some comments about fundamental cycle times and computation times. Since the F8 CPU has a maximum clock cycle of 2 MHz, at the clock the cycle time is 500 ns. Also, it has two kinds of machine cycles: 2 μs (4 clock cycles) and 3 μs (6 clock cycles). The range of the instruction cycle is 2 μs$\sim$13 μs. The rest time of each machine cycle in which the CPU is not accessing the RAM is used for refreshing the RAM or accessing by the DMA. The speed of transfer by the DMA is about 2 μs$\sim$ 5 μs per a byte.

The floating point numbers are all expressed in 4 byte length (one byte for exponent and three bytes for mantisa). In our subroutines of floating point arithmetic operations the computation times are as follows: the addition and subtraction takes the range of 26 μs$\sim$1600 μs. We take 1 ms as a typical value for a later evaluation. Multiplication by Booth's method takes about 3.5 ms and division by the non-restoring method takes about 4 ms. The transfer of 4 bytes between two registers takes about 20 μs and that between a register and a memory takes 32 μs. The transfer of 4 tybes through a FIFO memory takes 20 μs in a standard case, when data go through without staying. However, when many words are transfered continuously, the transfer time per one word is, of course, shorter. For example, when 16 words are transfered continuously, it is about

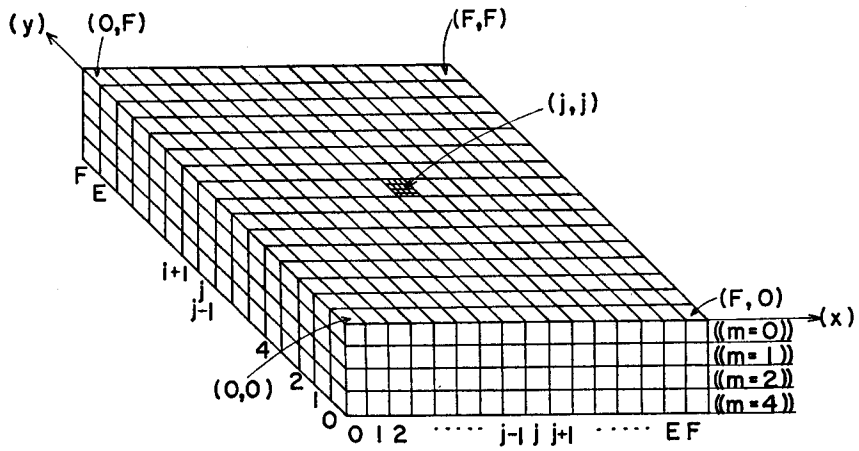

Fig. 4. Exclusive memories in the MU.

20F0 ~	21F0 ~	· · ·	2FF0 ~
20E0 ~	21E0 ~	· · ·	2FE0 ~
⋮	⋮	⋱	⋮
2010	2110	· · ·	2F10
2000	2100	· · ·	2F00

Fig. 5. Address allocation of the MU's exclusive memories.

2.5 μs. Such transfer operations take more time in addition for setting some data before the jump to subroutines. Their times are 17 μs, 25 μs and 53 μs before a transfer between two registers, between a register and a memory and a transfer through a FIFO respectively. The time for transferring through a FIFO also contains the setting time.

Finally, we provide some memory areas of exclusive use in the RAM for the convenience of making programs for a transfer through buffer memories. In the MU's RAM ,we take a two dimensional array of memories with a 16×16 size corresponding to the FIFO's array, and let each memory have a capacity of 4 words (16 bytes). We call it a memory block with 4 words depth, and give the words the numbers of $m=0\sim3$. (See Fig. 4). For the total memories of 4096 bytes we allot the part of the RAM with addresses from 2000 to 2FFF. The memory block with addresses $2ij0\sim2ijF$ corresponds to the node of a two dimensional array (i,j) $(i,j=0, 1, \cdots, F)$. (See Fig. 5).

On the other hand, in the AU-j's RAM, we take two one-dimensional arrays of memory blocks which correspond to the FIFO's ,directly connected with the AU-j by

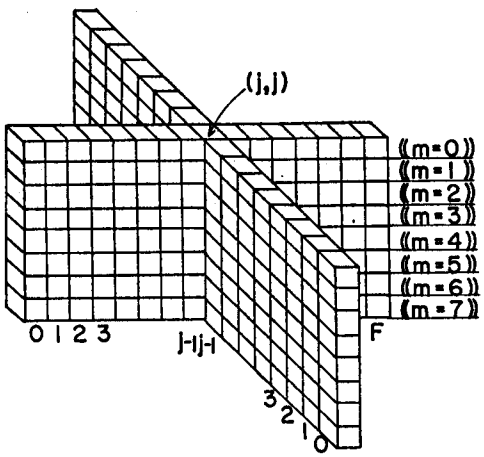

Fig. 6. Exclusive memories in the AU's.

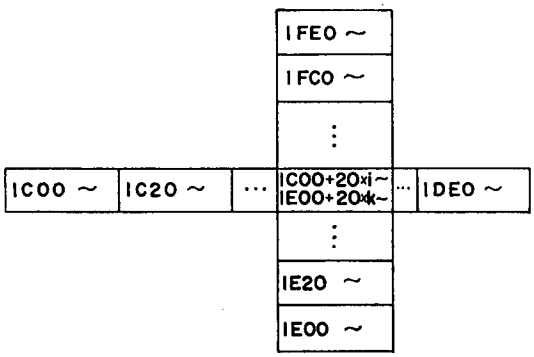

Fig. 7. Address allocation of the AU's exclusive memories.

buses. Each block has a capacity of 8 words. We number the words from $m=0$ to $m=7$ (Fig. 6). For these memories, we allot the part of the RAM with addresses from 1C00~1FFF. One half of them, with addresses 1C00~1DFF, correspond to the row of FIFO-(i, j) ($i=0, 1, \cdots, F$). The other half, with addresses 1E00~1FFF, correspond to the column of FIFO-(j, k) ($k=0, 1, \cdots, F$). The memory block having addresses $1C00+20\times i$~$1C1F+20\times i$ corresponds to the node (i, j), and that having addresses $1E00+20\times k$~$1F1F+20\times k$ corresponds to the node (j, k) (Fig. 7).

In the following, we denote sliced memories with the same level of depth by the symbol $((m=\cdot))$. For the AU's, we call the row part of the cross type array 'the memory row', and the other part 'the memory column'.

§ 4 Evaluation of efficiency in various computations

In this section, we evaluate the efficiency of the ADINA computer by various typical examples of computation. We will compare the time between a serial computation by the MU alone and a corresponding parallel computation by the AU's for the same problem. The comparison is fair since the LSI chips suitable for the AU's are the same as in the MU. The computation time is evaluated by following a program written in a machine language. Even though we use some subroutines which we can not evaluate exactly (for example, floating point operations), we believe that an evalaution using suitable mean times leads to a result close to the realized result. As indexes of comparison we use

Ratio of times $r=$ *Uni-processing time*$/$*Parallel processing time*

and, considering the number of processors,

Efficiency $R=r/N$.

Example 1 (Product of two matrices)

We suppose that the elements of two matrices A and B, with a size of 16×16, are

given in (($m=0$)) and (($m=1$)) of the MU's RAM respectively, and that the product $C=AB$ is asked in (($m=0$)).

In a case of computation by the AU's ,the MU first sends each row vector $\vec{a}_j=(a_{j0}, a_{j1}, \cdots, a_{jF})$ of A to each memory row (($m=0$)) of the AU-j through the FIFO-(j, j) for $j=0, 1, \cdots, F$, and then each column vector $\uparrow b_k=(b_{0k}, b_{1k}, \cdots, b_{Fk})^t$ to each FIFO-(k, i) ($i=0, 1, \cdots, F$) for $k=0, 1, \cdots, F$. Then every FIFO is filled up. Now, the AU-j ($j=0, 1, \cdots, F$) receives data from the FIFO-(k, j) ($j=0, 1, \cdots, F$) respectively, computes $\vec{a}_j \cdot \uparrow b_k=\sum_{j=0}^{F} a_{ji}b_{ik}$ and stores the results in the memory row (($m=1$)) successively for $k=0, 1, \cdots, F$. These computations are executed in a fully parallel way. Finally, each AU sends the data of the memory row (($m=1$)) to the MU's (($m=0$)).

The total time needed for the above computation is evaluated as 1.27 sec., and its essential part (computation of scalar product) is 1.20 sec.

On the other hand, we can evaluate the time needed in a case of uni-processing by the MU alone as 19.4 sec.

Therefore we get

$$r=15.3 \qquad \text{and} \qquad R=0.95.$$

Example 2 (The Gauss elimination method for solving a linear equation)

We solve a linear equation of 16 unknown numbers $\{x_i\}$, $i=0, 1, \cdots, F$, by the Gauss elimination method:

$$\sum_{i=0}^{F} a_{ji}x_i=b_j \qquad (j=0, 1, \cdots, F).$$

We suppose that the row vectors $\vec{a}_j=(a_{j0}, a_{j1}, \cdots, a_{jF})$ ($j=0, 1, \cdots, F$) of the coefficient matrix $A=\{a_{ij}\}$ are given in the memory row (($m=0$)) of the MU. The right hand side $\{b_j\}$ is also given somewhere in the RAM of the MU, and suppose that it is asked to obtain the solution $\{x_i\}$ in the same memories as $\{b_j\}$ is stored.

First, the MU sends b_j to the AU-j for $j=0, 1, \cdots, F$, successively. After that, the AU's proceed to solve the problem. In the following, we consider only a case where it is not necessary to make a special pivoting.

For eliminating x_0 the AU-0 first determines the coefficient

$$\left(\frac{a_{01}}{a_{00}}, \frac{a_{02}}{a_{00}}, \cdots, \frac{a_{0F}}{a_{00}}, \frac{b_0}{a_{00}}\right)$$

and sends it to each AU-j ($j=1, 2, \cdots, F$) through the FIFO-$(j, 0)$ ($j=1, 2, \cdots, F$) respectively. After receiving the coefficient, every AU, except the AU-0, eliminates x_0 in parallel. At the next step, for eliminating x_1, the AU-1 determines the corresponding coefficient and sends it to each AU-j ($j=2, 3, \cdots, F$), which then eliminates in parallel. Continuing the same procedure we finally arrive at a stage where the AU-F has an equa-

tion involving only one unknown x_F. After finding x_F, it is then substituted in all the equations that had successive elimination. Here, x_F is first sent to each AU-j ($j=E$, D, $\cdots$, 0) in the decreasing order of j successively. It is substituted on the corresponging processors. Next, x_E is found by AU-E, and it is sent to each AU-j ($j=D$, C, $\cdots$, 0) and then substituted. By continuing in this way we can find every x_j. Here we note that, while AU-j sent the found x_j to every AU-i ($i=j-1, j-2, \cdots, 0$), each AU-i can substitute x_j immediately after receiving it, without waiting for the completion of the AU's sending. In particular, the AU-$(j-1)$ can find x_{j-1} at once. Moreover each AU-j may send the found x_j to the MU immediately.

The total time for the above computation is evaluated as 887 ms. On the other hand, the time for uniprocessing by the MU alone is evaluated as 7277 ms. Therefore, we have

$$r=8.2 \qquad \text{and} \qquad R=0.51.$$

In this case, the fall to 0.51 of efficiency is because of an essentially serial property of the Gauss algorithm itself.

Example 3 (The FFT algorithm)

First, we briefly explain the FFT algorithm in order to show its relation to the ADINA-I. Now we define the discrete Fourier transformation by the formula

$$X(n)=\sum_{k=0}^{N} x_0(k)W^{kn}, \qquad W=e^{-i\frac{2\pi}{N}}.$$

Here we take $N=2^\gamma$ (γ is a natural number) for simplicity. In the FFT algorithm we perform the sequence of computation

$$\{x_0(k)\}\rightarrow\{x_1(k)\}\rightarrow\cdots\rightarrow\{x_l(k)\}\rightarrow\cdots\rightarrow\{x_\gamma(k)\}.$$

Finally, $X(n)$ is set to be equal to $x_\gamma(\bar{n})$, $\bar{n}$ means the binary number produced from the binary number n through reversing the order of digits. In order to get $x_l(k)$ from $x_{l-1}(k)$, we use the following formulae;

$$\left.\begin{aligned} &x_l(k)=x_{l-1}(k)+W^p x_{l-1}(k') \\ &\text{and} \\ &x_l(k')=x_{l-1}(k)-W^p x_{l-1}(k') \qquad (k'=k+N/2^l), \end{aligned}\right\} \tag{1}$$

where p is the binary number produced from the index k by expressing it in the form of γ digits, then shifting it to the right by $(\gamma-l)$ digits, putting zero's on the upper $(\gamma-l)$ digits and finally reversing the order of the digits.

We write the formula (1) in the following convenient form:

$$
\left.\begin{aligned}
&ARG=2\pi p/N,\\
&C=COS\,(ARG),\\
&S=SIN\,(ARG),\\
&k'=k+N/2^l,\\
&TR=Re\,x_{l-1}(k')\times C+Im\,x_{l-1}(k')\times S,\\
&TI=Im\,x_{l-1}(k')\times C-Re\,x_{l-1}(k')\times S,\\
&Re\,x_l(k')=Re\,x_{l-1}(k)-TR,\\
&Im\,x_l(k')=Im\,x_{l-1}(k)-TI,\\
&Re\,x_l(k)=Re\,x_{l-1}(k)+TR,\\
&Im\,x_l(k)=Im\,x_{l-1}(k)+TI.
\end{aligned}\right\} \qquad (2)
$$

In the above algorithm $x_l(k)$ and $x_l(k')$ are computed in pairs. Therefore, when we follow the algorithm by the MU alone for $k=0, 1, \cdots$, successively, we arrive at just one already computed. Then we, of course, may skip it. For example, in the case of $N=16$, ($\gamma=4$), $\{x_1(k)\}$ are all found by executing the algorithm (2) only for $k=0, 1, \cdots, 7$.

Now we state how to realize the algorithm by the AU's in as much parallel as possible. First, we consider the case in which a table of values being used for COS and SIN are given beforehand. Corresponding to a pair of numbers k and k', the AU-k and AU-k' execute (2) in parallel. First the AU-k' sends $x_{l-1}(k')$ to the AU-k through the FIFO-(k, k') and the AU-k sends $x_{l-1}(k)$ to the AU-k' through the FIFO-(k', k). Here both actions are taken simultaneously. Next, the AU-k' finds TR and sends it to the AU-k through the FIFO-(k, k'), and the AU-k finds TI and sends it to the AU-k' through the FIFO-(k', k). These actions are also taken simultaneously. Finally, the AU-k' finds $x_l(k')$ and the AU-k finds $x_l(k)$ at the same time. Therefore, we find that the computations also proceed in a fully parallel manner for all pairs of (k, k'). Such computations are executed for $l=1, 2, \cdots, \gamma$.

We consider also the case in which the values of COS and SIN are computed just when they are asked. In this case it is convenient to use the CORDIC method for the evaluation of COS and SIN, because it is well realized again by each pair of the AU-k and AU-k' (See the next example for the CORDIC method itself.)

In a case where the 16 AU's are used for the data of $N=16$, the total time is evaluated as follows:

i) (The case of having a table of COS and SIN's)

The serial computation by the MU alone takes 690 ms. On the other hand it takes 60 ms that the MU first gives data to the AU's and then they run and send results to the MU. Therefore we get

$$r=11.5 \qquad \text{and} \qquad R=0.72.$$

ii) (The case of using the CORDIC method)

The serial computation by the MU alone takes 2600 ms, while the parallel computation by the AU's takes 244 ms. Therefore,

$$r=10.66 \quad \text{and} \quad R=0.67.$$

The fall to 0.72 of efficiency in case (i) is because the computation by the AU's proceeds so well in paris, but it takes time for transfering data, while the computation by the MU can skip one half of the set of numbers $\{k\}$. In case (ii), the efficiency further falls because the CORDIC method is not reduced to a fully parallel computation.

Example 4 (Evaluation of COS and SIN by the CORDIC method)

In Example 3, it was necessary to have the values of $\mathrm{COS}(p\pi/8)$ and $\mathrm{SIN}(p\pi/8)$ $(0\leqq p\leqq 7)$. Here we consider the CORDIC method for evaluating them by using a pair of processors AU-k and AU-k'. We note that we have to evaluate them only for $0\leqq p\leqq 3$ since $\mathrm{COS}(p\pi/8)=-\mathrm{COS}((8-p)\pi/8)$ and $\mathrm{SIN}(p\pi/8)=\mathrm{SIN}((8-p)\pi/8)$. Then for $0\leqq p\leqq 3$, the method is as follows;

We take

$$x_0=0.60725293051\cdots, \quad y_0=0 \quad \text{and} \quad v_0=p\pi/8$$

as the initial values, and use the following formula for $n=0, 1, \cdots, 16$ iteratively:

i) if $v_n\geqq 0$, $x_{n+1}=x_n-2^{-n}y_n$, $y_{n+1}=y_n+2^{-n}x_n$, $v_{n+1}=v_n-\gamma_n$

ii) if $v_n<0$, $x_{n+1}=x_n+2^{-n}y_n$, $y_{n+1}=y_n-2^{-n}x_n$, $v_{n+1}=v_n+\gamma_n$

where $\gamma_n=\arctan 2^{-n}$ $(n=0, 1, \cdots, 16)$ are given veforehand. The result x_{17} and y_{17} are approximate values of $\mathrm{COS}(p\pi/8)$ and $\mathrm{SIN}(p\pi/8)$, respectively, with an accuracy at least by five digits of decimal numbers.

Let each AU-k and AU-k' have the initial values and $\{\gamma_n\}$ and evaluate doubly the same $\{v_n\}$. Then, let the AU-k find $\{x_n\}$ and the AU-k' find $\{y_n\}$.

For computing with a p it takes 47.7 ms. On the other hand, in the case of serial computation by the MU alone it is 60 ms. Therefore

$$r=1.2 \quad \text{and} \quad R=0.6.$$

Since the part of computing $\{v_n\}$ is not executed in parallel by both processors, the fall of efficiency is inevitable.

Example 5 (Application of the ADI method for solving the Navier-Stokes equation)

The two-dimensional flow of non-compressible viscous fluid is described by the following Navier-Stokes equation

$$\left.\begin{aligned}&\frac{\partial\zeta}{\partial t}+\frac{\partial\psi}{\partial y}\frac{\partial\zeta}{\partial x}-\frac{\partial\psi}{\partial x}\frac{\partial\zeta}{\partial y}=\frac{1}{R}\Delta\zeta \quad (R \text{ is the Reynolds number.}),\\&\Delta\psi=-\zeta\end{aligned}\right\}$$

where ζ represents the vorticity and ψ is the stream function.

Now we consider a flow in the quadratic hollow region $0<x,y<1$ with only one opened side $y=1$ contacting with a uniform flow. Then we have the following natural boundary conditions:

$$\psi(0,y,t)=\psi(1,y,t)=\psi(x,0,t)=\psi(x,1,t)=0,$$
$$\frac{\partial\psi}{\partial x}(0,y,t)=\frac{\partial\psi}{\partial x}(1,y,t)=\frac{\partial\psi}{\partial y}(x,0,t)=0$$

and

$$\frac{\partial\psi}{\partial y}(x,1,t)=U \qquad (U \text{ is the constant velocity of the uniform flow.})$$

The initial conditions are

$$\psi(x,y,0)=0 \qquad \text{and} \qquad \zeta(x,y,0)=0$$

corresponding to the stationary state.

We take the most useful ADI method from many difference schemes for solving the above problem. We divide each side of the hollow region into 15 equal parts, and cover the region by the net of mesh points with the mesh width $h=1/15$. We use an expression like $\zeta_{i,j}$ showing the value of the unknown ζ at the mesh point $(x_i, y_j)=(ih, jh)$. Taking the time step τ, we develope it like $\zeta^n_{i,j}$ which means the value at $t=n\tau$. For finite differences, we use the following notations;

$$\Delta_1\zeta_{i,j}=\zeta_{i+1,j}-2\zeta_{i,j}+\zeta_{i-1,j}, \qquad \Delta_2\zeta_{i,j}=\zeta_{i,j+1}-2\zeta_{i,j}+\zeta_{i,j-1},$$
$$\delta_1\zeta_{i,j}=\zeta_{i+1,j}-\zeta_{i-1,j}, \qquad \delta_2\zeta_{i,j}=\zeta_{i,j+1}-\zeta_{i,j-1}.$$

In the following algorithm, we take $J=15$;

$$\left.\begin{aligned}&\zeta^0_{i,j}=\psi^0_{i,j}=0 \qquad (i,j=0,1,\cdots,J),\\&\zeta^{n+1/2}_{i,j}-\frac{\tau}{2Rh^2}\Delta_1\zeta^{n+1/2}_{i,j}+\frac{\tau}{8h^2}\delta_2\psi^n_{i,j}\delta_1\zeta^{n+1/2}_{i,j}=A^n_{i,j}\\&A^n_{i,j}=\zeta^n_{i,j}+\frac{\tau}{2Rh^2}\Delta_2\zeta^n_{i,j}+\frac{\tau}{8h^2}\delta_1\psi^n_{i,j}\delta_2\zeta^n_{i,j}\\&\qquad\qquad (i=1,2,\cdots,J-1),\\&\zeta^{n+1/2}_{0,j}=-\frac{2}{h^2}\psi^n_{1,j}, \qquad \zeta^{n+1/2}_{J,j}=-\frac{2}{h^2}\psi^n_{J-1,j}\\&\qquad\qquad (j=1,2,\cdots,J-1)\end{aligned}\right\} \quad (1)$$

$$\left.\begin{aligned}&\zeta^{n+1}_{j,k}-\frac{\tau}{2Rh^2}\Delta_2\zeta^{n+1}_{j,k}-\frac{\tau}{8h^2}\delta_1\psi^n_{j,k}\delta_2\zeta^{n+1}_{j,k}\\&\qquad=2\zeta^{n+1/2}_{j,k}-A^n_{j,k} \qquad (k=1,2,\cdots,J-1),\end{aligned}\right\} \quad (2)$$

$$\zeta_{j,0}^{n+1}=-\frac{2}{h^2}\psi_{j,1}^{n} \qquad \zeta_{i,j}^{n+1}=-\frac{2}{h^2}\psi_{j,J-1}^{n}-\frac{2}{h}U$$

$$(j=1, 2, \cdots, J-1)$$

$$\psi_{i,j}^{(0)}=\psi_{i,j}^{n} \qquad (i, j=0, 1, \cdots, J),$$

$$\psi_{i,j}^{(r+1/2)}-\frac{\sigma}{h^2}\Delta_1\psi_{i,j}^{(r+1/2)}=B_{i,j}^{(r)},$$

$$B_{i,j}^{(r)}=\psi_{i,j}^{(r)}+\frac{\sigma}{h^2}\Delta_2\psi_{i,j}^{(r)}+\sigma\zeta_{i,j}^{n+1} \qquad (3)$$

$$(i=1, 2, \cdots, J-1),$$

$$\psi_{0,j}^{(r+1/2)}=\psi_{J,j}^{(r+1/2)}=0 \qquad (j=1, 2, \cdots, J-1)$$

$$\psi_{j,k}^{(r+1)}-\frac{\sigma}{h^2}\Delta_2\psi_{j,k}^{(r+1)}=2\psi_{j,k}^{(r+1/2)}-B_{j,k}^{(r)}$$

$$(k=1, 2, \cdots, J-1), \qquad (4)$$

$$\psi_{j,0}^{(r+1)}=\psi_{j,J}^{(r+1)}=0 \qquad (j=1, 2, \cdots, J-1)$$

$$(r=0, 1, \cdots, P_{n+1}-1)$$

$$\psi_{i,j}^{n+1}=\psi_{i,j}^{(P_{n+1})} \qquad (i, j=0, 1, \cdots, J)$$

$$(n=0, 1, 2, \cdots)$$

The above formulae give the algorithm for proceeding from the n-th time step to the $(n+1)$-th step. The upper suffix (r) of ψ means the number of the internal iteration, P_{n+1} is the total number of the iteration at the $(n+1)$-th step and σ is an iteration parameter.

A realization of the above algorithm by the 14 AU's is as follows:

The first half of the ζ-equation (1) is composed of the 14 equations which are linear in the unknown vectors $\vec{\zeta}_j^{n+1/2}=(\zeta_{1,j}^{n+1/2}, \zeta_{2,j}^{n+1/2}, \cdots, \zeta_{J-1,j}^{n+1/2})$ $(j=1, 2, \cdots, J-1)$. Let each AU-j solve the j-th equation. The second half of the ζ-equation (2) is also composed of the 14 equations in the unknown vectors $\uparrow\zeta_j^{n+1}=(\zeta_{j,1}^{n+1}, \zeta_{j,2}^{n+1}, \cdots, \zeta_{j,J-1}^{n+1})^t$. Let each AU-$j$ also solve its j-th equation. Then both (1) and (2) are solved in fully parallel. We take also a similar way for solving the linear equations (3) and (4) of $\psi^{(r+1/2)}$ and $\psi^{(r+1)}$, respectively.

Now, we suppose that every AU-j $(j=1, 2, \cdots, J-1)$ has the data $\vec{\zeta}_j^{n}=(\zeta_{1,j}^{n}, \zeta_{2,j}^{n}, \cdots, \zeta_{J-1,j}^{n})$, $\uparrow\zeta_j^{n}=(\zeta_{j,1}^{n}, \zeta_{j,2}^{n}, \cdots, \zeta_{j,J-1}^{n})^t$ and $\uparrow\psi_j^{n}=(\psi_{j,1}^{n}, \psi_{j,2}^{n}, \cdots, \psi_{j,J-1}^{n})^t$ in their own memories. Further, we suppose that every AU-i $(i=1, 2, \cdots, J-1)$ has $\Delta_2\uparrow\zeta_i^{n}=(\Delta_2\zeta_{i,1}^{n}, \Delta_2\zeta_{i,2}^{n}, \cdots, \Delta_2\zeta_{i,J-1}^{n})^t$, $\delta_2\uparrow\zeta_i^{n}=(\delta_2\zeta_{i,1}^{n}, \delta_2\zeta_{i,2}^{n}, \cdots, \delta_2\zeta_{i,J-1}^{n})^t$ and $\delta_2\uparrow\psi_i^{n}=(\delta_2\psi_{i,1}^{n}, \delta_2\psi_{i,2}^{n}, \cdots, \delta_{i,J-1}^{n})^t$ in their memor column. It should be here mentioned that the AU-i cannot find $\Delta_1\zeta_{i,j}^{n}$ or $\delta_1\zeta_{i,j}^{n}$ etc. if it has only column vectors $\uparrow\zeta_i^{n}$ etc., while it can find $\Delta_2\uparrow\zeta_i^{n}$ and $\delta_2\uparrow\zeta_i^{n}$ easily.

Now the procedure is as follows:

i) First the AU-j receives $\Delta_2\zeta_{i,j}^{n}$, $\delta_2\zeta_{i,j}^{n}$ and $\delta_2\psi_{i,j}^{n}$ from each AU-i $(i=1, 2, \cdots,$

$J-1$) throughth FIFO-(i, j)'s respectively. It then has the equation (1) in $\vec{\zeta}_j^{n+1/2}$ with the unknown coefficients, and on the right hand $\vec{A}_j^n=(A_{1,j}^n, A_{2,j}^n, \cdots, A_{J-1,j}^n)$. Since A_j^n is again used later for solving (2), it is held in the memory. Morevoer, at this stage, the AU-j finds $\frac{\tau}{8h^2}\delta_1\vec{\psi}_j^n$ and stores it.

ii) The AU-j then solves the equation in $\vec{\zeta}_{i,j}^{n+1/2}$ by the so-called double sweep method. In the process of $\{\zeta_{i,j}^{n+1/2}\}$'s being successively determined in the inverse sweep order, the AU-j sends the triple $\{\zeta_{i,j}^{n+1/2}, \frac{\tau}{8h^2}\delta_1\psi_{i,j}^n, A_{i,j}^n\}$ one by one to the corresponding FIFO-(i, j), without waiting until $\vec{\zeta}_j^{n+1/2}$ is fully found.

iii) Next, the AU-j receives all the triples $\{\zeta_{j,k}^{n+1/2}, \frac{\tau}{8h^2}\delta_1\psi_{j,k}^n, A_{j,k}^n\}$ through the FIFO-(j, k) ($k=1, 2, \cdots, J-1$) respectively. Then, the AU-j knows all the coefficients and the right hand of the equation (2) in $\vec{\zeta}_j^{n+1}$, solves the equation and sends the pairs $\{\zeta_{j,k}^{n+1}, \Delta_2\psi_{j,k}^n\}$ to the FIFO-(j, k) ($k=1, 2, \cdots, J-1$) respectively. (Here we suppose, of course, that $\{\Delta_2\psi_{j,k}^n\}$ has been already found.)

iv) The AU-j receives the pairs $\{\zeta_{i,j}^{n+1}, \Delta_2\psi_{i,j}^n\}$ from the FIFO-(i, j) ($i=1, 2, \cdots, J-1$) respectively. Combining them with $\vec{\psi}_j$ being stored, the AU-j has the equation (3) with the known right hand $\vec{B}_j^{(0)}$, then solves it and sends the pairs $\{B_{i,j}^{(0)}, \psi_{i,j}^{(1/2)}\}$ to the FIFO-(i, j) ($i=1, 2, \cdots, J-1$) respectively.

v) The AU-j receives the pairs $\{B_{j,k}^{(0)}, \psi_{j,k}^{(1/2)}\}$ from the FIFO-(j, k) ($k=1, 2, \cdots, J-1$) respectively. It then finds $\uparrow\psi_j^{(1)}$ from the equation (4) and sends the pairs $\{\Delta_2\psi_{j,k}^{(1)}, \psi_{j,k}^{(1)}\}$ to the FIFO-(j, k) ($k=1, 2, \cdots, J-1$) respectively.

vi) The AU-j receives the pairs $\{\Delta_2\psi_{i,j}^{(1)}, \psi_{i,j}^{(1)}\}$ and finds $\vec{B}_j^{(1)}$. By solving (3) the AU-j gets $\vec{\psi}_j^{(3/2)}$ and again sends the pairs $\{B_{i,j}^{(1)}, \psi_{i,j}^{(3/2)}\}$ to the FIFO-(i, j) ($i=1, 2, \cdots, J-1$) respectively.

vii) Repeating the procedure for $r=0, 1, \cdots, P_{n+1}-1$ like v) $\rightarrow$ vi) $\rightarrow$ v) $\rightarrow \cdots \rightarrow$ vi) $\rightarrow$ v) finally produces $\{\psi_{j,k}^{(P_{n+1})}=\psi_{j,k}^{(n+1)}\}$ ($j, k=1, 2, \cdots, J-1$). Only in the last procedure of v), the AU-j sends $\{\zeta_{j,k}^{n+1}, \Delta_2\zeta_{j,k}^{n+1}, \delta_2\zeta_{j,k}^{n+1}, \delta_2\psi_{j,k}^{n+1}, \psi_{j,k}^{n+1}\}$ to the FIFO-(j, k) ($k=1, 2, \cdots, J-1$) respectively. Then, the computation of the next time step starts again from i).

As seen above, several sets of data must be sent or held. For that, we use the exclusive memories in the AU's. (See Fig. 6–7.) It is shown in Fig. 8 how data are shared in the memories, and which data are sent to and received from the FIFO's. The row with the heading H-R contains data that the AU-j receives and stores in the memory row. The row with 'H-S' contains data which the AU-j sends from its memory row to the FIFO's. Similarly, the row with 'V-R' contains data which the AU-j reseives and stores in its memory column, and that with 'V-S' forms that which it sends from its memory column. It completes the one time step from the n-th to the $(n+1)$-th to take the stages 'H-R, H-S' $\rightarrow$ 'V-R, V-S' $\rightarrow$ 'H-R, H-S' $\rightarrow \cdots \rightarrow$ 'V-R, V-S' from top to bottom. Here it is shown that the values not shadowed in the sending states (S)and the receiving states (R) must be held for a while, and are already held respectively.

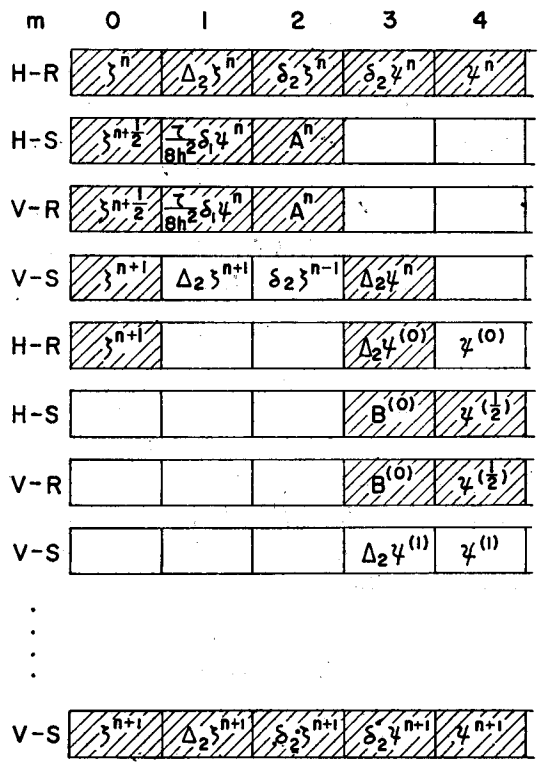

Fig. 8. Data allocation in a memory block in Example 5.

The computation time for proceeding by one time step is estimated as 3875 ms in the case of $P_{n+1}=5$. On the other hand, when the MU alone executes the same procedure, it is 54156 ms. Therefore

$$r=13.8 \quad \text{and} \quad R=13.8/14=0.99.$$

The result shows a very high efficiency. It is because the rate of transferring data through a FIFO memory is about 18 μs/word. The summing up transferring times in above computation accounts for only 0.5 percent of the total time, and the essential part of computation is almost parallel.

Here we note that we used only 14 AU's, but we can use, of course, all 16 AU's if we expand the exclusive memories beforehand.

We can generalize the above result and give an estimate in the case of using Q AU's. Denoting the number of internal iteration by P, we have

$$(74.1+40.3P)Q+1.0P+15.0 \text{ ms}$$

$$(275.6Q+20.0 \text{ ms} \quad \text{when} \quad P=5).$$

For a case by the MU alone, it is

$$(73.1+39.9P)Q^2+(16.2+0.2P)Q \text{ ms}$$
$$(272.6Q^2+17.2Q \text{ ms} \quad \text{when} \quad P=5).$$

Here we consider that the problem to be solved has $Q\times Q$ internal mesh points. Therefore we get for the large Q

$$(74.1+40.3P)Q \text{ ms} \quad \text{by the AU's}$$

and

$$(73.1+39.9P)Q^2 \text{ ms} \quad \text{by the MU alone.}$$

Thus

$$r=0.988Q \quad \text{and} \quad R=0.988 \quad \text{when} \quad P=1,$$
$$r=0.99Q \quad \text{and} \quad R=0.99 \quad \text{when} \quad P \text{ is also large.}$$

Glancing at the result, it seems that the high efficiency is due to such a complexity in the essential part of the computation for solving the nonlinear partial equation, that the time of transferring data is comparatively small. However, such efficiency is also produced in a case where the equations to be solved are themselves as simple as seen in the case of the Poisson equation, which is considered in the next example.

Example 6 (Application of the ADI method for solving the Poisson equation)

The second half of the Navier-Stokes equation in Example $\Delta\psi=-\zeta$ is nothing but the Poisson equation. Therefore in order to estimate, we have only to consider the part of internal iteration for finding ψ.

In the computation by the AU's it is 2822 ms when the number of iteration is 5, while in the computation by the MU alone it is 39095 ms. Therefore

$$r=13.85 \quad \text{and} \quad R=0.99.$$

Example 7 (A simple lteration method for solving the Poisson equation)

Here we consider again the Poisson equation $\Delta\psi=-\zeta$, and solve its difference analogue by the following simple iteration method:

$$\psi_{i,j}^{r+1}=\psi_{i,j}^{r}+\frac{\sigma}{h^2}(\psi_{i+1,j}^{r}+\psi_{i-1,j}^{r}+\psi_{i,j+1}^{r}+\psi_{i,j-1}^{r}-4\psi_{i,j}^{r})+\sigma\zeta_{i,j}$$
$$(r=0, 1, 2, \cdots).$$

In order to realize the method we repeat the following procedures:

i) The AU-j receives the sums $\{\psi_{i,j+1}^{r}+\psi_{i,j-1}^{r}\}$ from the FIFO-(i, j) $(i=1, 2, \cdots, J-1)$ respectively. Finding $\{\psi_{i,j}^{r+1}\}$ by the above formula, the AU-j sends them to the FIFO-(i, j) $(i=1, 2, \cdots, J-1)$ respectively.

ii) The AU-j receives $\{\psi_{j,k}^{r+1}\}$ through the FIFO-(j, k) $(k=1, 2, \cdots, J-1)$ respectively, finds $\{\psi_{j,k+1}^{r+1}+\psi_{j,k-1}^{r+1}\}$ and sends them to the FIFO-(j, k) $(k=1, 2, \cdots, J-1)$ respectively.

It takes 992 ms for a 5 time iteration, while it taker 13469 ms for a computation by the MU alone. Therefore

$$r=13.58 \quad \text{and} \quad R=0.97.$$

In general, when the number of AU's is Q and the number of iterations is P it is 14.1 QP ms, while in a case by the MU alone, it is 13.7 Q^2P ms. Therefore

$$r=0.97\,Q \quad \text{and} \quad P=0.97.$$

This also shows a very high efficiency.

Example 8 (The Gauss-Seidel method for solving the Poisson equation)

We consider the standard Gauss-Seidel method for solving the same problem as in Examples 6 and 7, which is described as

$$\psi_{i,j}^{r+1}=\frac{1}{4}(\psi_{i-1,j}^{r+1}+\psi_{i+1,j}^{r}+\psi_{i,j-1}^{r+1}+\psi_{i,j+1}^{r})+h^2\zeta_{i,j} \qquad (r=0,\ 1,\ 2,\ \cdots).$$

In order to realize this method, we have the AU-j play the role of finding $\vec{\psi}_j^{r+1}$. Here we notice that when the AU-2 asks, for example, $\psi_{1,2}^{r+1}$ it must also know $\psi_{2,1}^{r+1}$ which is to be found by the AU-1. Therefore, when the AU-1 finds $\psi_{2,1}^{r+1}$, it is better to send it to the AU-2 through the FIFO-(2, 1), and then continue to find $\psi_{3,1}^{r+1}$. Thus, $\{\psi_{i,j}^{r+1}\}$ is successively determined as a wave in the order of the number $i+j$'s increasing.

The AU-j's action is as follows: the AU-j receives $\psi_{i,j+1}^{r}$ from the FIFO-$(j, j+1)$, $\psi_{i,j-1}^{r}$ from the FIFO-$(j, j-1)$, then finds $\psi_{i,j}^{r+1}$, which it sends to the FIFO-$(j-1, j)$ and FIFO-$(j+1, j)$ for each i. After the completion of such computations, it continues to find $\vec{\psi}_j^{r+2}$. Here it is noticed that only the FIFO's on the tri-diagonal are used.

When the number of AU's is Q and that of iterations is P, it takes 8.6 $(P+1)$ Q ms, while in a case by the MU alone it is $(7.8\,Q^2+0.05\,Q)P$ ms. Therefore,

$$r=(0.907\,Q+0.0058)\left(1+\frac{1}{P}\right)^{-1}.$$

When Q and P are large,

$$r=0.907\,Q \quad \text{and} \quad R=0.907.$$

the result shows a slight fall of efficiency in comparison with that of the ADI method. The reason is that the Gauss-Seidel method is primarily a serial method, and is not realized in fully parallel.

As seen in Examples 5, 6, 7 and 8, the ADINA computer shows a very high efficiency

of 0.9~0.99, when it solves the partial difference schemes by some iteration methods. Especially, by the ADI method, the computation speed is almost Q (Q is the number of AU's.) times that of the corresponding serial computation.

Acknowledgement

It will take some months before completion of the first trial manufacture of the ADINA computer. We express our appreciation to Dr. A. Matsumura, Miss Y. Nakagawa, Mr. R. Kobayashi, Mr. S. Otanaka, Mr. H. Ohwaki and Mr. Y. Inoue for their help in the construction of hardware and software.