ADENA Computer IV

AUTHOR(S):

NOGI, Tatsuo

CITATION:

NOGI, Tatsuo. ADENA Computer IV. Memoirs of the Faculty of Engineering, Kyoto University 1993, 55(1): 21-36

ISSUE DATE:

1993-01-29

URL:

http://hdl.handle.net/2433/281470

Mem. Fac. Eng., Kyto Univ., Vol. 55, No. 1 (1993)

ADENA Computer IV

By

Tatsuo NOGI*

(Received September 30, 1992)

Abstract

A new parallel computer system, ADENA IV, is proposed as a most hopeful candidate for the next generation machine for supercomputing. It is a complex of vector processor units and ADENA's network and may be called a vector-parallel computer. Details of the architecture are exposed in relation to its usage, and an example of compact design is shown as a model for personal supercomputing. ADENA IV will succeed the present supercomputers of the vector type as well as ADENA II and III, and it promises to have merits in relation to both present highend techniques. Its parallel language also may succeed ADETRAN for the ADENA family.

1. Introduction

Large scientific simulations usually demand both much memory and high speed. To supply that, some supercomputers of the vecotr type and parallel computers of the multiporocessor type have been developed in last two decades [1,2]. Since the vector computer itself almost reaches the acme of development, it is expected that some systems with a number of vector processors or a lot of scalar processors or ones of the mixed type will form the next generation of supercomputers. Today we already have systems with several vector processors and a vast shared storage of many memory banks, but they are only an extension of conventional vector machines and rely on the idea of dividing a simple long vector into several parts to be computed by many pipelines. On the other hand, current parallel computers of the multiprocessor type mostly use the distributed memory system, and they use anyone of three types of network to transfer data among processors: the grid, the hyper-cube and the hyper-cross-network type. The first has been used from the early stage, but it is not so hopeful due to problems in data transfer ability. The second, the hyper-cube network, is now most appreciated in the U.S., and it supplies more data transferring ability than the first. But its whole system generally assumes an inhomogeneous scheme of transfer, which cannot help

* Applied Systems Scinece, Kyoto University, Kyoto, 606-01, Japan.

reducing its perofrmance. The last, the *hyper-cross-network*, is very powerful in allowing uniform data gransfer among processors and therefore giving us a simple scheme of data passing. It was proposed by the author [3–9] and we already have it in a practical system with very high power [10–11]. That system with distributed storage and network, however, produces more or less overhead of data transfer, which diminishes its performance.

This paper proposes a new system, which we call a *vector-parallel computer*, with many vector processors (or alternatively, procesosr arrays) and data memory banks shared among those processors in an organized but restricted way. This new machine is characteristic not only in having a variety of access routes to the data storage but also in practicing some data edition in the storage itself as a whole system. *Data edition* was one of the most fundamental concepts in early ADENA systems I and II [3–5], and it stood for general data transmission. The concept continues to live in the new system ADENA IV, but the array of buffer memories on the hyper-corss-network is now replaced by the array of banks, which allows logical data edition by only changing access ways to the memory array without the need to practice real data transmission. Actually, the last model of ADENA's family, ADENA III [8] has replaced buffer memories placed on all nodes of the 3–dimensional hyper-corss-network in ADENA II with memory banks, to remove the middle stage of buffering for edition, and has replaced the plane array of processors with distributed local storages by an array of vector processors (or processor arrays), which are able to access the memory bank system in the same way in two different directions as the processor array of ADENA II does to the hyper-cross-network cube. The last scheme is very useful for the necessary data edition, but it holds th same data array in such multiple ways as to waste extra memory. ADENA IV is now designed to have a scheme of three access ways (and one more way) to memory banks in order to solve the extra-memory problem. It practices necessary data edition without wasting extra storage. However, increasing access routes generally demand many additional bus lines, and hence it is essential to use as many bus lines in common as possible without increasing time for the data-transferring. This paper will offer a solution to this problem.

2. Outline of ADENA IV

The computer system of ADENA IV is also a hierarchy of a host computer (front-end processor) and a slave system for parallel processing as the former ADENAs. The slave system is essential and consists of three parts: 1) a processor part

which contains a control processor unit CU and a number N of vector processor units, say VU1, VU2,...,VUN, 2) a main memory part of many banks which are arranged in a cubic grid (N×N×N) and are joined by the sets of row and coulmn common busses that open access routes from vector processors, and 3) a vector-latch part of N vectors with length N (actually a two-diemnsional array of element-latches) standing between the first two parts.

Control unit CU has a program memory and a decoder of object codes. It sends control signals to vector units, the vector-latch part and the main memory part. Each vector unit VU has a number of vector pipelines and registers for processing vector data taken from the main memory, as well as a scalar processor and a scalar data memory and an input/output data channel from/to the front-end processor.

The vector-latch part has as many *vector-latches* VL1, VL2,...,VLN as VU's, each being connected with a corresponding VU. Each vector latch VL consists of N *element-latches*, and it has *parallel ports* for sending/receiving data on respective element-latches to/from the main memory part and two *serial ports*, one also being connected to the main memory part and the other to a corresponding vector register. Totally, those VL1, VL2,... and VLN are connected with the vector processing units through respective *serial busses* on one side, and with the main memory part through serial busses and *parallel common busses* on the other side. Here 'common' means that each parallel bus touches all element-latches forming a section of all vector-latches, as a whole.

The whole cubic array of memory banks are placed on N boards, with each board having a sliced subarray of N×N banks. Each bank has a temporary storage of N word length on its gateway, we call a *access-latch*. A set of row busses of the number N and another set of column busses of the same number run on every board as a lattice, and access-latches are placed on all cross poinst (nodes) so that they may be accessed through row or column busses joining at their respective nodes.

It is essential to allow four ways to access the main memory, and their characteristics are seen mainly in connection schemes between the main memory and the vector-latch part. They are explained in the next section.

3. Vector-parallel processing

We first clarify concepts of row, vertical and column vectors. We suppose to compute some three-diemnsional data produced originally in a coordinate

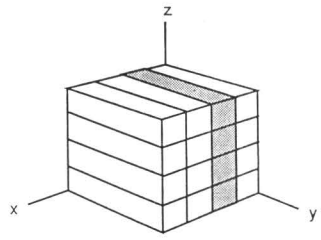

Fig. 1 Row vectors

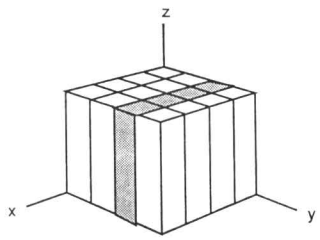

Fig. 2 Vertical vectors

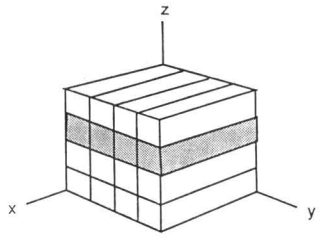

Fig. 3 Column vectors

system (x,y,z). We call a one-dimensional subarray with any specified z-and x-index a *row vector*, one with any specified x- and y-index a *vertical vector* and one with any specified y- and z-index a *column vector*. We then have three points of view for the same three-dimensional array, that is, the first is to see the whole array as a set of row vectors (Fig. 1), the second is as that of vertical vectors (Fig. 2) and the third is as that of column vectors (Fig. 3).

We then introduce four ways to access those vectors.

1) The first way is to access row vectors with all elements at the respective tops of the access-latches (which, in turn, here and in the following two cases correspond to data having the same local-address in banks) with any specified row number over all boards through row busses which are opened to the corresponding column's bus on respective boards and serial busses of the vector-latches, so that all those row vectors may be sent/receievd to/from the vector-latches.

2) We can consider a data set of elements having the same z-index, each taken from vertical vectors with the same column number. We call it a *column section*. The second way is to access column sections having all elements at the respective tops of the access-latches with the sepcified column number over all boards through column busses which are opened to parallel busses of vector-latches, so that all column sections of the same column number may be sent/received to/from the vector-latches. It then has all vertical vectors of the same column number. Consequently, all those vertical vectors can be sent/received to/from the vector-lathces.

3) The third way is to access column vectors with all elements at the respective tops of access-latches on the selected board through column busses which are opened to serial busses of vector-latches, so that all column vector data on that board may be sent/received to/from the vector-latches.

4) The final way is to access all the elements in access-latches (which correspond to data having a same regular sequence of local addresses in banks) with the specified row and column number passing through row busses which are opened

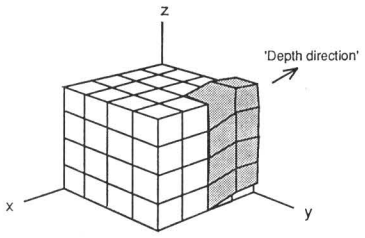

Fig. 4 Depth vectors

as in the third way.

We also have four kinds of parallel processing, corresponding to three points of view and an auxiliary point of view for 2–dimensional data arrrays: say, *processing in the x-, y-, z- direction* and an *auxiliary processing in the x-direction.* Processing in the x-direction is to compute all row-vectors with a specified x-index in vector-processing along the y-index and in parallel over the z-index, and to repeat it successively changing the x-index. Processing in the y-direction is to compute all vertical-vectors with a specified y-index in vector-processing along the z-index and in parallel over the x-index and to repeat it successively changing the y-index. Processing in the z-direction is to compute all column vectors with a specified z-index in vector processing along the x-index and in parallel over the y-index and to repeat it successively changing the z-index. Those processing ways use the respective access ways of 1), 2) and 3) above.

In addition, we also have one more processing mode which is realized by using access way 4). It appears in processing two-dimensional arrays, as explained later. Those arrays are mapped into four-dimensional (three and 'depth') arrays which are actually realized on the three-dimensionla architecture(Fig. 4). It stands for an auxiliary processing in the x-direction in the three-dimensional case, or we may call it *processing in the depth direction.* It is to compute all vectors with a specified x-and y-index on all boards formed by ranging over the depth index, in vector-processing along the depth index and in parallel over the z-index, and to repeat it successively changing the x- and y-indexes.

4. Details of Architecture

Here we give details of the system architecture, especially how access ways mentioned in the last section are realized. For illustration, we assume the number N=4. Interpretation may apply to more general cases with more processors, too.

As shown in Fig. 5, the system consists of the processor part, the main memory part and the vector-latch part. In the main memory part, there are 4 boards in

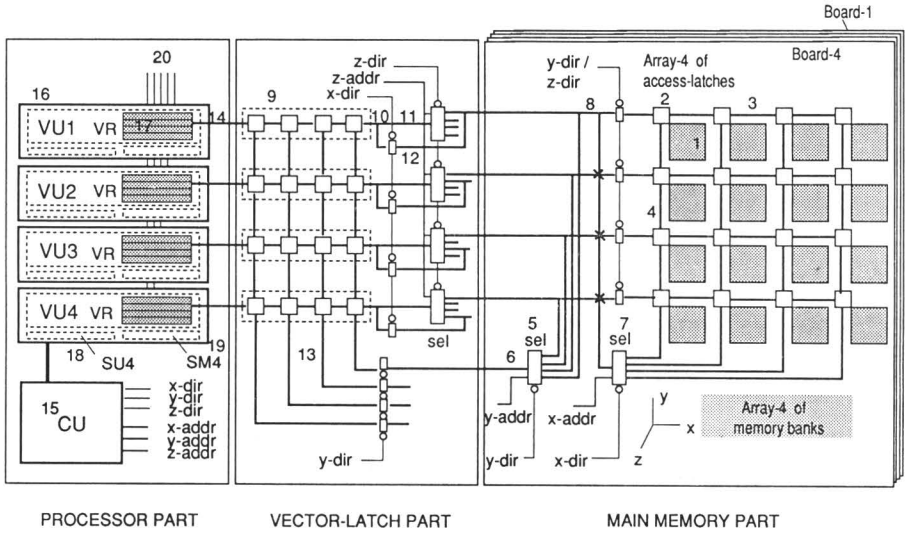

Fig. 5 Architecture

concert with N=4, each of which is assumed to be selected by specifying a z-index in a three-dimensional array, and has a two-dimensional subarray (N×N) of memory banks. Every bank (1) has a access-latch (2) of N word length which is placed on a corresponding cross point (node) of a grid network of column (3) and row (4) busses running in the x- and y-directions respectively. Those busses are common for all access-latches just on those busses. Column busses (3) on all

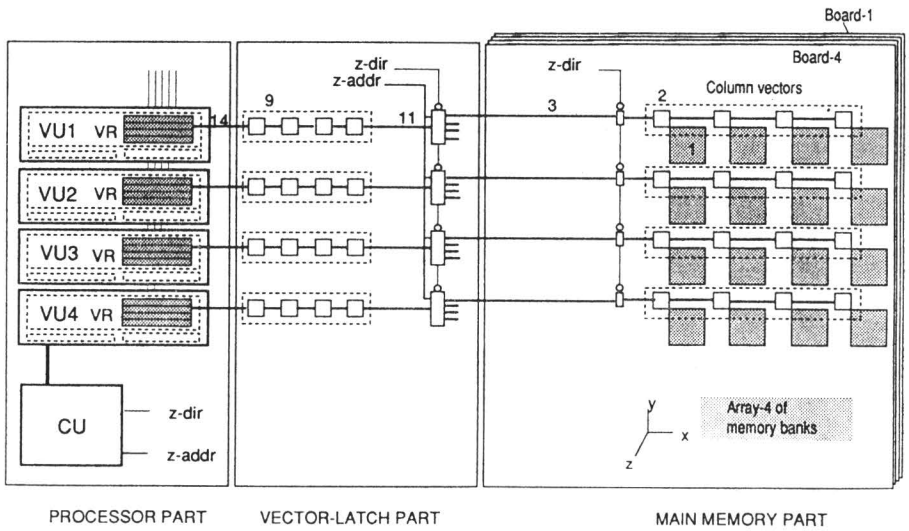

Fig. 6 Processing in the z-direction

boards are used in processing both in the z- and y-directions. In processing in the z-direction, they are directly joined to the serial busses of the vector-latches through a selector choosing one set of column busses from a specified board (see Fig. 6). In processing in the y-direction, column busses specified by the y-index (5), each of which we call the y-specifying-bus (6), are connected to the respective sections of vector-latches through their parallel busses (13) (see Fig. 7). In processing in the x-direction, row busses specified by the x-index, are connected to the column busses specified by the z-indexes, each of which we call the z-specifying-bus (8), and further are joined to the respective vector-latches (see Fig. 8).

The vector-latch part stands between the main memory and the processor part and temporarily holds a set of vectors of N word length. It has vector-latches (9) whose number is N, each of which has N element-latches. Serial busses (10) are placed on both sides of all vector-latches, facing the main memory and the processor part. In processing in the z-direction, serial busses on the side facing the main memory are connected to all column busses (11) on the board selected by the z-index, say, z-specifying-columns. In processing in the x-direction, those serial busses are connected to the respective z-specifying-column busses (12) and further x-specifying-row busses on all boards. In processing in the z- and x-directions, row- or column-vecotrs in the main momory themselves are transferred to/from the vector-latches through serial busses, while in processing in the y-direction, section data whose elements are those of the same z-index from the necessary ver-

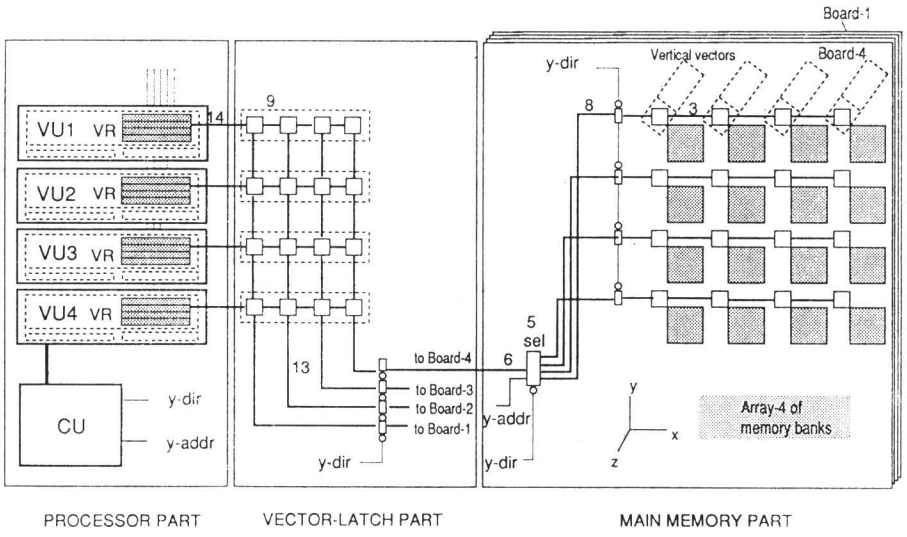

Fig. 7 Processidg in the y-direction

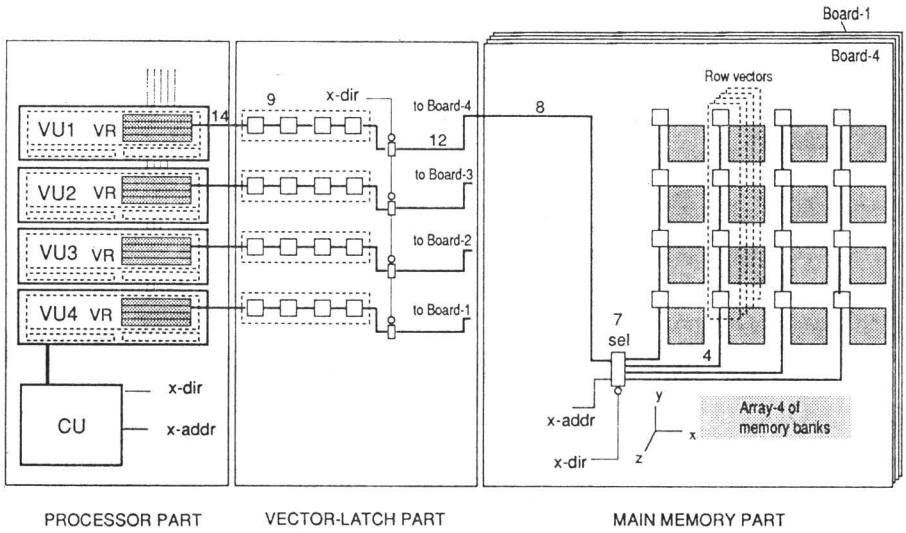

Fig. 8 Processing in the x-direction

tical vectors with a specified y-index are transferred to/from the vector-latches through parallel busses (13) which are common to all vector-latches, each parallel bus being connected to all element-latches of the same order number in the vector-latches. Those parallel busses are connected to y-specifying-busses (6) from memory boards. In processing in every direction, serial busses on the side facing the processor part (14) are directly connected to vector units.

The processor part has a control unit CU(15) and vector units (16) of the number N. Those vector units have the same configuration, a number of vector registers and another number of processing pipelines (17). Those word lengths are several times N so as to increase the performance of the vector processing. Every vector unit further has a scalar processing unit (18) and a scalar data storage (19). Those vector units have data channels (20) for high speed input/output of a vast number of data. The control unit gives a sequence of processing commands to all vector units, a sequence of access signals and addresses to the main memory part, and other control signals to the whole system. It also contains an instruction storage.

5. Standard scheme of computation

We here explain a standard scheme of computation. Suppose we are going to practice porcessing in the x-direction. When CU decodes a 'reading' code, it

sends the the x-direction signal, x-address and local address within banks. Then all data appear in the row of the top elements of the access-latches specified by the x-address, one column to one z-address. They are carried through the row busses, x-specifying-row busses and vector-latches to the vector registers. On changing local address, another set of column vectors from the same column of banks may be carried through the samt route as above to be appended to the vector registers. Continuing this process, we can get as long vectors as possible, which are transferred into pipelines for processing. It is important for such repetition to send data continuously through access-latches, vector-latches and vector registers. In order to do that, as soon as a part of a vector is sent to a next stage, its next part should appear. Especially, for the multiple use of banks to provide a virtual processor environment, several elements must be placed in the respective access-latches successively to read/write from/in banks, and the hardware should then give support for successive accesses. Processing in the y- or z-directions takes similar actions. Only a case of auxiliary processing in the x-direction is different from other cases in that vectors themselves are latched in access-latches in the depth-direction (see Fig. 9).

Processing in the x-direction easily computes the whole set of two row vector operands with different x-indexes or depth addresses in banks over a given range of y- and z-indexes, with an vector mode for the y-index and an parallel mode for the z-index. Similarly, processing in the y-(or z-) direction computes that of two

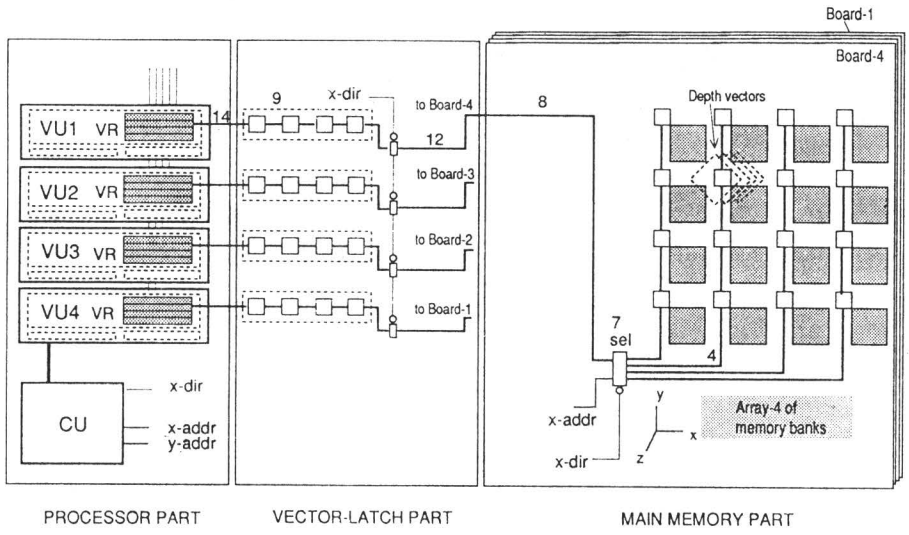

Fig. 9 Auxiliary processing in the x-direction

vertical (row) vector operands with different y-(z-) indexes or depth addresses over a given range of z-(x-) and x- (y-) indexes, with an vector mode for the z-(x-) index and an parallel mode for x-(y-) index. Further, auziliary processing in the x-direction for 4-dimensional array data computes that of two depth vector operands being different in the x- or y-index or in position within banks over a given range of the z-index, with an vector mode for the depth address and an parallel mode for the z-index.

6. Usage of the system

Our system is designed especially for 3- or 2-dimensional simulations, and matrix computations in scientific and engineering problems. It just fits the processing of 3–dimensional array data well. Taking such processing as a fundamental one, we expand and apply it for 2–dimensional and matrix problems.

First we will expain the fundamental 3–dimensional scheme. Its characteristic is easily seen in the way how to put a 3–dimensional data, say $\{u(i,j,k), i=1, 2, \ldots, pN, j=1, 2, \ldots, qN, k=1, 2, \ldots, rN\}$, into the main banks of memory: in putting $i=(P-1)N+I, j=(Q-1)\ N+J$, and $k=(R-1)N+K$, each element $u(i,j,k)$ is placed at the $\{(R-1)\,pq+(Q-1)p+P\}$th position after that of the first element $u(I,J,K)$ in the (I,J,K)th bank. This means that (I,J,K) selects one of the banks and (P,Q,R) determines a position in the selected bank. Every bank is occupied by the number pqr of elements. It may suggest the ability to access N^3 elements with the same (P,Q,R), at one time. That bandwidth is, however, too huge to implement in reality, and our system has only a bandwidth of one-order-less so as to access N^2 elements at one time, which are specified by fixing one of I,J, and K and running the remaining indexes.

The next characteristic is to allow any one of three access ways as desired. Such ways are distinguished by the expression of elements using indexes enclosed by slashes as seen in the following examples:

(1) $u(i,|j,k|)$ (2) $u(i|,j,|k)$ (3) $u(|i,j|,k)$

The first is to access any 2–dimensional subarray (section) with a specified incex i, ranging over the region of j and $k, j=1, 2, \ldots, qN, k=1, 2, \ldots, rN$. It is needless to say that those accesses of N^2 elements must be repeated qr times for $Q=1, 2, \ldots, q$ and $R=1, 2, \ldots, r$ to complete full access. This way is just for processing in the x-direction. The second is to access any section with a specified index j, ranging over $k=1, 2, \ldots, rN$ and $i=1, 2, \ldots, pN$. This corresponds to processing in the y-

direction. Finally, the third is to access any section with a specified index k, ranging over i and j. It must be noted that those expressions denote only different access ways, but mean the same data. If it would be wanted to distinguish between vector and parallel processing, we could express the array as

$$(1)\quad u(i, /j, /k//) \qquad (2)\quad u(/i//, j, /k) \qquad (3)\quad u(/i, /j//, k)\,.$$

The first expression (1) means that necessary processing is serial in i and parallel in j and k, with vector processing for j which is enclosed singly by slashes and with purely parallel processing for k which is enclosed doubly by slashes. Similarly, the second (2) (third (3)) means that it is serial in $j(k)$, vector in $k(i)$ and purely parallel in $i(j)$. It is ,however, assumed that both vector and purely parallel processing are categorized only in parallel processing and users need not know such distinctions in a well devised programming environment.

For example, consider a simple FORTRAN program:

```
      do 10 k = 1, 16
        do 10 j = 1, 16
          do 10 i = 2, 15
10          v(i, j, k) = u(i+1, j, k) +u(i-1, j, k)
```

It may be written as

```
      pdo j = 1, 16, k = 1, 16
        do 10 i = 2, 15
10          v(i, /j, k/) = u(i+1, /j, k/) +u(i-1, /j, k/)
      pend
```

Here, we used only single slashes because there is no necessity of expressing distinctions between voctor and parallel processing and for simplicity.

As seen in the above example, we may use only such indexes in slashes in a *pdo-pend* clause as listed just in the *pdo* statement, and may not use their general expressions or other variables in slashes. The reason is as follows: index k, which would be enclosed doubly in slashes, is just for purely parallel processing, and correspond to the order number of vector units. It leads to the necessity of data transfer among vector units to allow scmething like expressions, but may conflict with the simplicity of the *pdo-pend* clause for parallel processing. On the other hand, about index j which would be enclosed singly by slashes, such simple expressions as $j+1$ or $j-1$ might be allowed because they would require only the shift operations in vector registers, but to prohibit more general expressions it is better to suppose no allowance also for j as for k. These restrictions produce no problems since changing

the direction of processing allows general expression in a corresponding bare index part.

We are going to the 2–dimensional problems. It is also important to solve those problems effectively, since 2–dimensional simulations themselves appear in many applications and matrix problems come from many branches as their final step to solution. The essential point is how a 2–dimensional array, say $\{u(i,j), i=1, 2, \ldots, N^2, j=1, 2, \ldots, N^2\}$, should be mapped into the memory banks of a 3–dimensional grid configuration. We must here have two ways of processing, which are expressed in the data array as

$$(1)\quad u(i, /j/) \qquad (2)\quad u(/i/, j)$$

The former means processing in serial for the bare index i and in parallel for the slashed index j(processing in the x-direction), and the later menas processing in serial for the bare index j and in parallel for the slashed index i(processing in the y-direction). Whichever expression is taken, they are the same data. A solution of the mapping problem is the following: first assign index pairs, (r,q) and (t, s), for the original indexes i and j respectively. They are put in a relation of

$$i = (q-1)N+r, \qquad j = (s-1)\ N+t.$$

Both forms of the 2–dimensional array may be mapped to 4–dimensional arrays

$$(1)\quad u(r, /q, /t//)\ (s) \qquad (2)\quad u(/r, /q//, t)\ (s)$$

where the triple r,q and t specify any one of 3–dimensional memory banks and the index s selects respective local positions in the banks. Clearly the former is for processing in the x-direction in a 3–dimensional case and the latter is for processing in the z-direction. We mention here that in (2), the original slashed index $/i/$ itself is replaced by the slashed $/r,/q//$, which means that once j or (t,s) is specified, the data of number N^2 may be accessed in parallel for i or (r,q), while in (1), the index i specifies not only the bare r but also the slashed q, which means that instead of N^2, only data of the number N may be accessed in parallel for t. In order to deal with data of N^2 for vector-parallel processing, it is necessary to also access data of number N in all banks with index (q,t). To realize it, it is better to have memory banks further divided into so many sub-banks to allow the so-called interleave way of access, or to use such fast devices as the so-called RDRAM (Rambus Inc. USA) or Synchronous DRAM (USA JEDEC). Those data of length N with continuous s's may be transferred through x-specifying row busses and serial busses to/from vector registers. This is just the auxiliary processing in the x-direction. Data

for such processing might be exactly written as

$$(1') \quad u(r, q, //t//)\ (/s/)$$

This is the expression not seen in 3–dimensional cases. Only the system software is concerned with such realization, and users have only to understand two ways of processing for $u(i, /j/)$ and $u(/i/, j)$. For example, to get the sum of two matrices $\{a(/i/, j), i, j=1, 12, \ldots, 256\}$ and $\{b(/i/, j), i, j=1, 2, \ldots, 256\}$, we have only to write the following program:

```
pdo  i = 1,256
  do  10  j = 1,256
10    a(/i/,j) = a(/i/,j)+b(/i/,j)
pend
```

The system sortware will expand it as follows:

```
pdo  r, q = 1,16
  do  10  t = 1,16
    do  10  s = 1,16
10        a(/r, /q//, t) (s) = a(/r, /q//, t) (s)+b(/r, /q//, t) (s)
pend
```

On the other hand, for $\{a(i, /j/)\}$ and $\{b(i, /j/)\}$, the program

```
pdo  j = 1,256
  do  10  i = 1,256
10    a(i, /j/) = a(i, /j/)+b(i, /j/)
pend
```

will be expanded as

```
pdo  t, s = 1,16
  do  10  r = 1,16
    do  10  q = 1,16
10        a(r, q, //t//) (/s/) = a(r, q, //t//) (/s/)+b(r, q, //t//) (/s/)
pend
```

Here we have shown some styles of writing programs based upon ADETRAN [9]. We must notice that ADETRAN may be transfered to present ADENA IV without much modification.

7. An example of implementation

We will here propose a compact system of using today's available vector chips. Suppose $N=16$. The main memory banks are placed on 16 boards. The processor part consists of 16 other boards for vector units and one board for the control unit. Those boards all are pluged in a mother board which also has the vector latch part on itself.

Every memory board has a bank array of size 16×16. Every bank consists of 16 chips of 4Mbit ($1M\times 4$) DRAM(50*ns*), 1M—64bitW, so that every board has 4096 chips, 256MW(2048MB) and the whole system has 4096MW(32GB). Every bank is attached by a access latch of 64bit-width and 16 word-length. One set of 16 64bit-busses runs as rows and the other set of 16 64bit-busses runs as columns on each obard.

Every board has 1088 data-bus lines, i.e. $1024=64\times 16$ from column busses and 64 from row busses on its edge. The 32 address-bus linea are sufficient to appear on its edge, and the remaining lines are for two control signals to determine processing direction and clock, etc.

The 32bit address is necessary for 4096MW. The 32 bits are divided as follows: in a 3-dimensional problem, any one of coordinates is assigned 8 bits, 4 bits of which are for a physical bank address and the other 4 bits are for a logical/virtual bank address. The remaining 8 bits are for identifying variables. At most, the system allows the solution of those problems of 256 variables on a $256\times 256\times 256$ grid region; in a 2–dimensional problem, 4–dimensional arrays are used. The 6 bits are supplied for 3 directions, with 4 bits being for a physical bank address and 2 bits for a virtual bank address. The remaining 14 bits are for addressing in each bank, with 6 bits for the fourth dimension and the others for identification. It means that we can solve problems with up to 256 variables on a 4096×4096 mesh region.

The vector-latch part has 16 vector-latches of 16 word length, which are connected to the main memory part and the processor part through serial busses, and also to the main memory part through parallel busses. Memory access through those vector latches is based upon the interleave way; the memory cycle time 48*ns*/16W is sustained, including delay of latches and selector circuits.

The processor part has 16 vector chips of CMOS runing on 100 MHz. Every chip has 16 vector registers of 256 64bit-word length and 4 pipelines for vector processing, and has a buffer for commands, an internal control and bus unit, and scalar unit together with scalar registers. This chip is attached with an external

storage for scalar data, 4096W.

The control unit also has a processor of 100MHz and instruction memory (1 MW).

Its peak speed is about 6.4GFLOPS, while its sustained speed in a situation of continuous operation of only 2 pipelines may be 3.2GFLOPS, which realizes about 1.7W/FLOP in memory throughput.

8. Conclusion

ADENA IV replaced the buffer memories placed at all cross point nodes of the hyper-cross-network as seen in ADENA II with the memory banks on those nodes, which constitute the main storage and allow three standard ways and an auxiliary way to access them. Those ways assure the full ability of ADENA II with no overhead due to data transfer and hence allow to get a very high sustained performance.

Today's supercomputers with several vector units are supplied with a common memory array of banks, usually being accessible from all vector units, but they colud not have many more vector units because it would become hard to secure sufficient access ways to many banks without bank conflict. On the other hand, ADENA IV allowed vector units only to access partial banks in the regular scheme from the outset. The acheme is based upon the concept of PSSS (Parallel over Segments and Serial in Segment) [11] and ADE (Alternating Direction Edition) common st the whole family of ADENAs. We think that the scheme is necessary and sufficient for practicing scientific computations, and that it is not too narrow to access. We consider such a concept to be essential for parallel processing, and we hope it will be accepted also by other future systems. By the way, we mention about ADENA IV that 'Edition' in ADE is no longer necessary and ADE should now mean Alternating Direction Execution.

Finally we may say that a new generation of supercomputers will come from systematizing vector units and memory units well enough to compute in parallel as far effectively as possible, and its most hopeful answer is ADENA IV.

References

1) K. Hwang and F.A. Briggs: Computer Architecture and Parallel Processing, McGraw-Hill (1985).

7) K. Hwang and D. Degroot, editors: Parallel Processing for Supercomputers & Artificial Intelligence, McGraw-Hill (1989).

3) T. Nogi and M. Kubo: ADINA Computer I, I. Architecture and Theoretical Estimates, Mem.

Fac. Eng. Kyoto Univ. 42 (4) (1980) 421–439.
4) T. Nogi: ADINA Computer II, I. Architecture and Theoretical Estimates, ibid, 43 (1) (1981) 124–144.
5) T. Nogi: ADINA Computer I and II, II. Data Structure, ibid, 43 (3) (1981) 434–450.
6) T. Nogi: Parallel Machine ADINA, in 'Computing Methods in Applied Sciences and Engineering V', eds. R. Glowinsky and J.L. Lions, North-Holland (1982) 103–122.
7) T. Nogi: Parallel Computation, in Patterns and Waves, Studies in Mathematics and its Applications 18, Kinokuniya/North-Holland, (1986), 279–318.
8) T. Nogi: ADENA Computer III, Mem. Fac. Eng. Kyoto Univ. 51 (2) (1989) 135–152.
9) T. Nogi: Parallel Programming Language ADETRAN, Mem. Fac. Eng. Kyoto Univ. 51 (4) (1989) 235–289.
10) H. Kadota, K. Kaneko, Y. Tanikawa and T. Nogi: VLSI Parallel Computer with Data Transfer Network: ADENA, Proc. 1989 Int. Conf. Parallel Processing, I 319–322.
11) T. Nogi: Parallel Computation on ADENA, Parallel Computing' 91, D.J. Evans et al. editors (1992) 619–626.