QCDPAX - An MIMD array of vector processors for the numerical simulation of quantum chromodynamics

Tomonori SHIRAKAWA[a], Tsutomu HOSHINO[a], Yoshio OYANAGI[b], Yoichi IWASAKI[c], Tomoteru YOSHIE[c], Kazuyuki KANAYA[c], Shingo ICHII[d] and Toshio KAWAI[e]

[a] *Institute of Engineering Mechanics,* [b] *Institute of Information Sciences* and
[c] *Institute of Physics, University of Tsukuba, Tsukuba, Ibaraki 305, JAPAN*
[d] *National Laboratory for High Energy Physics, Tsukuba, Ibaraki 305, JAPAN*
[e] *Department of Physics, Keio University, Hiyoshi, Kouhoku-ku, Yokohama 223, JAPAN*

1. INTRODUCTION

The lattice gauge theory is a discrete model of quantum chromodynamics(QCD), and it is considered to be able to calculate the strong interaction physical observables from the first principle by the numerical simulation of this model. This model is defined on the four-dimensional hypercubic lattice with the periodic boundary condition. Quark field represented by a 12 component complex vector is allocated on each node of the lattice, and gluon field represented by a 3x3 complex matrix is allocated on each link of the lattice. By this huge degrees of freedom, the simulation of QCD requires huge amount of floating-point operations. The quenched (ignoring the effect of quark field) simulations on the 16^3x48 lattice typically take 1000 hours of computing time on supercomputers of several hundred MFLOPS. We would like to obtain a factor 100 more of computer resources in order to get results at the level of a few percent statistical and systematic errors.

However it is easy to see that the simulation of lattice gauge theories is suitable to the parallel processing. Because it has high degree of freedom, homogeneity, and locality. Exploiting these features allows us to construct the specialized processor array of high performance and low cost. There are similar projects to construct parallel computers specially designed for the QCD in Columbia University, IBM, and Italy[1]-[3].

QCDPAX is a processor array designed for the simulation of the lattice QCD. PAX is the name of the series of the parallel computer since 1977 for the study of parallel high-speed computation in scientific or engineering applications, and four models (PAX-9, PAX-32, PAX-128, and PAX-64J) have been built so far[4]-[9]. PAX utilizes the MIMD processor array architecture with the two-dimensional nearest-neighbor connection and the broadcasting bus. Through the experiments on the PAX computers, it has been proved that the architecture of PAX is effective not only for the continuum simulation but also for the applications which have not the nearest-neighbor structure. For example, ADI schemes, FACR(Fourier Analysis and Cyclic Reduction) schemes, Gauss-Jordan method, and conjugate gradient method are demonstrated on the PAX computers. In the field of quantum physics, Monte-carlo simulation of U(1) model on four-dimensional lattice was completed on the PAX-128, and lattice QCD of 4^3x8 lattice was simulated on the PAX-64J.

QCDPAX is the fifth model in the PAX series [10]. A prototype with four processing units was constructed in the April 1988, and a practical system with 288 processing units was built in the April 1989.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1989 ACM 089791-341-8/89/0011/0495 $1.50

2. CALCULATION OF LATTICE QCD

The procedure of the calculation of lattice QCD is as follows. The gluon fields are to be updated according to the probability proportional to

$$\exp(-B\,S_p(U))\det(D(U)) \qquad (1)$$

where B is a constant and S_p is given by $1-\mathrm{Re}\ \mathrm{tr}[U_{12}U_{23}U_{34}U_{41}]/3$. U_{12} is a 3x3 special unitary matrix and represents the gluon field of the link which connects the nodes 1 and 2 etc. Configurations are generated by Langevin or hybrid method and physical quantities like Wilson loops, Polyakov loops, hadron propagaters are averaged over the configurations.

Dominant calculation in lattice QCD is to solve the linear equation

$$Dx = b \qquad (2)$$

where D is a sparse complex matrix. The size of the matrix D is 12 x number of nodes. This linear equation is solved by the minimum residual method with incomplete LU factorization preconditioning[11].

The essential calculation of the minimum residual method is the multiplications of a vector by a matrix. As the non-zero elements in the matrix appear only at the positions corresponding to the nearest neighbor interactions, the products of a vector and a matrix can be calculated in terms of the values of the nearest neighbor links.

The preconditioning $(LU)^{-1}y=U^{-1}L^{-1}y$ consists of a forward substitution and a backward substitution. These substitutions are essentially sequential processes. However once the substitution to a element is completed, parallel processing to the four neighboring elements can be started.

A number of physical quantities are calculated on the configurations. One of them is the Wilson loops, which is the trace of the products of the 3x3 matrices lying along the closed loop which is made by the concatenation of links. This calculation needs the value of non-nearest-neighbor variables. Another one is the propagation of quarks, which is obtained by the solution of equation (2).

The features of lattice QCD calculation are summarized to the following three points.

(1) It is the simulation of the interactions on the four-dimensional lattice with periodic boundary conditions. The interactions mainly occur between the nearest-neighbor nodes and links, and the reference of the data in distance is not often.
(2) It is a Monte-Calro simulation using the large amount of random number.
(3) It needs huge amount of 3x3 complex matrices calculations.

3. ARCHITECTURE OF THE QCDPAX

(1) TWO-DIMENSIONAL NEAREST-NEIGHBOR MESH CONNECTION

As the lattice gauge model is defined on the four-dimensional hypercube lattice, it would be natural to construct a parallel computer of four-dimensional nearest-neighbor mesh architecture. However, the four-dimensional connection requires eight communication paths for each processing unit and makes it difficult to keep the wide bandwidth of each communication path. Four-dimensional shape is difficult to build in the real space which has only three dimensions. Therefore QCDPAX adopts the two-dimensional nearest-neighbor mesh connection like the former PAX computers.

Two of the four dimensions of the lattice gauge model are mapped onto the two-dimensional PU array, and the rests are treated as an array in a PU. By this mapping, the neighboring nodes are mapped either in the same PU or the neighboring PU.

The incomplete LU preconditioning on the QCDPAX starts from one element and spreads out to the neighboring elements through the nearest-neighbor connections in a wave front fashion. This method, called hyperplane method, is a fully parallel process except for a few tens steps of its beginning and ending.

(2) MIMD ARCHITECTURE

QCDPAX utilizes MIMD architecture for the following reasons. Machine clocks in MIMD processor array need not be synchronized among the processing units. This is the great advantage of the MIMD architecture when the number of processing units is large.

The lattice gauge theory is a uniform model, and the programs of the PU's are the same with one another. But the execution flow of the program depends on the random numbers generated in each PU, and differs with each other.

The hyperplane method for the parallel processing of incomplete LU preconditioning is easier to install in the MIMD than SIMD.

(3) HIGH SPEED FLOATING POINT OPERATION SYSTEM FOR EACH PU

The simulation of lattice QCD requires huge amount of floating point operations such as the multiplication of complex matrices and random number generation. The each PU of the QCDPAX has the specialized high speed floating point operation system.

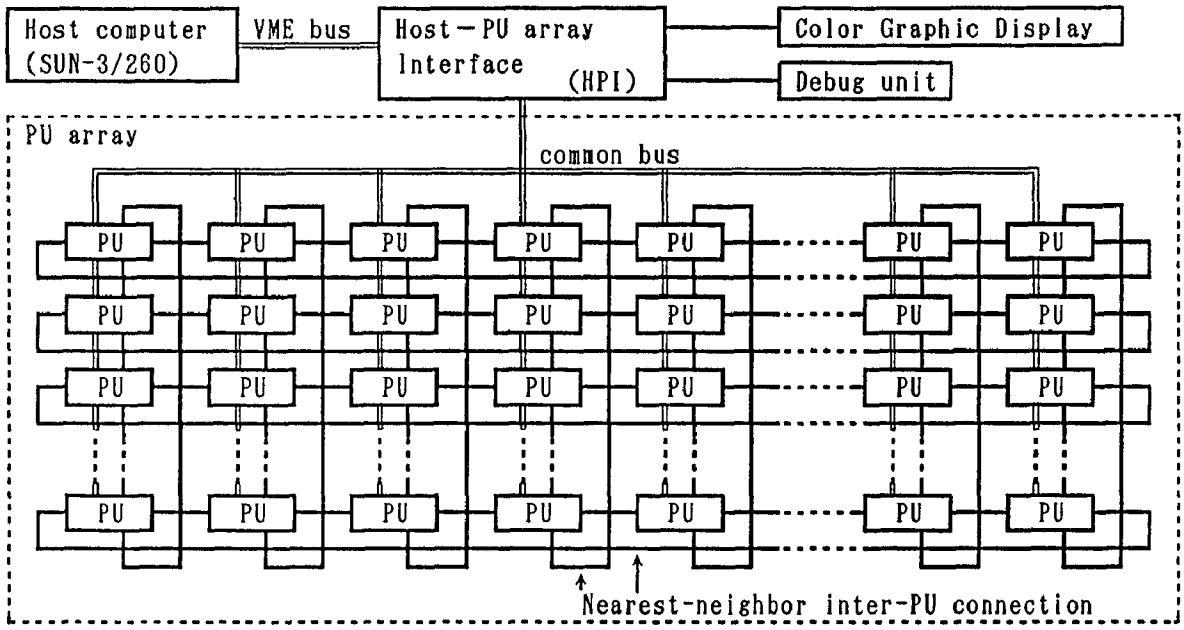
4. HARDWARE OF QCDPAX

4.1 SYSTEM CONFIGURATION

The system configuration of QCDPAX is shown in Fig.1. QCDPAX system consists of an array of processing units(PU array), a host-computer, a graphic display, and the interface between them (HPI).

The PU array is the hardware which executes the parallel tasks. The PU array works as the back-end processor of the host-computer. A PU shares a two-port RAM with the each of its four nearest-neighbors as shown in Fig.1.

PU's on the edge are connected to those on the opposite edge to form a torus. A common bus connects all PU's and the HPI. The memory



PU : processing unit.

Fig.1. System configuration of the QCDPAX.

of each PU is mapped on the address space of host-computer through the HPI, and works as an intelligent memory. HPI serves to select which PU(s) is mapped on the host-computer's address space. As HPI can map all PU's by overlapping them, the HPI can broadcast the program and the data to all PU's.

The host-computer is used to compile and assemble the source program, load the object program into the PU array, initiate parallel tasks, transfer and receive the data to and from the PU array. Sun-3/260 is used as the host-computer.

QCDPAX outputs the computational results from each PU through the two port RAM and the common bus. HPI is connected to the graphic display with a video processor. HPI collects the computational results (which may be changed into video image by the PU's) from the PU's and output to the display. This facility realizes to display the progressing computational process in the PU's such as the time evolution schemes.

The debug probe card is a card for debugging the parallel software, which can be connected to the bus of any PU. It picks up the signal on the inner-PU bus and reports the informations for debugging to the HPI.

4.2 PROCESSING UNIT (PU)

Each processing unit(PU) is an independent one-board microcomputer. The configuration of a PU is shown in Fig.2. Each PU essentially consists of a microprocessor(CPU), local memory, three communication memories (CM), two communication port, one synchronization register, a bus-interface between the PU bus and the common bus, a port for debug probe, a timer, a high-speed floating-point operation system, and the PU control circuit.

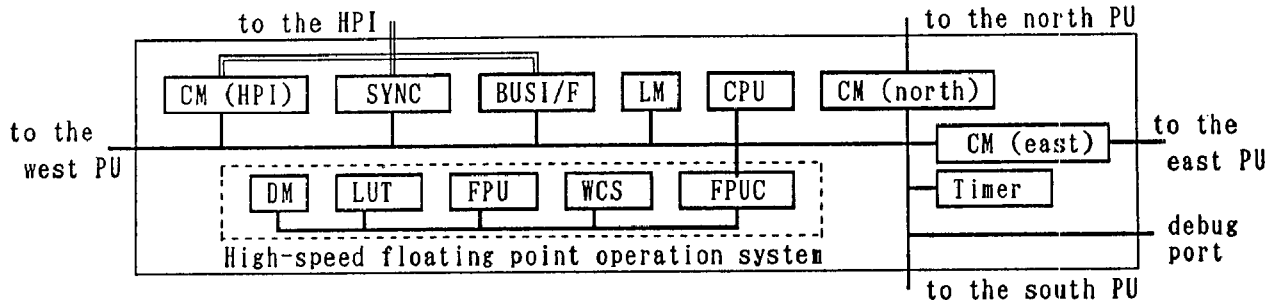
Motorola's microprocessor MC68020(25MHz) is used as the CPU. The local memory is 4MBytes with 100ns 1Mbit DRAM. Three wait states are inserted at the access from the CPU.

A communication memory is a 32bit x 2Kwords two-port RAM. Two CMs are used for the communication to the four nearest-neighbor PU's. The other CM is used for the communication with HPI/host-computer. Four 8bit x 2KBytes 90ns ready-made two-port RAM including arbitration hardware (Fujitsu's MB8421/8431) are used for one CM. At least two wait states are inserted when the CPU accesses the CM. The arbitration may stretch the wait state of the CM longer.

The synchronization register (SYNC) is a register to take the synchronization of all PU's. The output of the SYNC is connected to the HPI and the SYNC of the other PU's through the open collector common bus. The procedure of the synchronization is; (1)Each PU writes a code to the SYNC and stops; (2)If HPI detects the coincidence of the contents of all SYNCs, let all the PU to continue. Width of the SYNC is decided to be 6bit to take 64 kinds of synchronization.

The two-port RAM between HPI and PU is used for the communication of the HPI/host-computer and the PU during the execution of the PU task.

The host-computer/HPI can reset the PU and halt the CPU. The host-computer can access to



CM : Communication memory. (8KB 2portRAM).
SYNC : Syncronization register.
BUSI/F: Bus interface.
LM : Local memory. (4MB DRAM).
CPU : Central processing unit. (MC68020 25MHz).
DM : Data memory. (2MB High speed SRAM).
LUT : Look up table. (1KB ROM).
FPU : Floating Point operation Unit. (L64133 : Max.32MFLOPS).
WCS : Writable control strage. (8KB High speed SRAM).
FPUC : FPU controler.

Fig.2. Configuration of a single processing unit.

the inner-PU facilities, when the CPU of the PU is halting.

Each PU includes a timer in order to evaluate the efficiency of the QCDPAX in the applications. The resolution of the timer is 80ns and the size of the counter is 40bit.

4.3 HIGH-SPEED FLOATING-POINT OPERATION SYSTEM

The QCD calculation needs huge amount of complex computation, each PU is designed to have the capacity of high speed floating-point operation. The co-processor sold with the microprocessor is easy to interface with the microprocessor, but its computation speed is not fast enough. The QCDPAX utilizes the floating-point processing unit(FPU) L64133 on the market. L64133 has peak performance of the 33MFLOPS. It comprises an arithmetic logic unit and a multiplier concurrently workable. This feature is suitable to the complex calculation, for the operation of the complex numbers allows parallel execution of addition or subtraction and multiplication. The FPUC (floating-point processing unit controller), newly developed by the gate array, is also utilized to derive the performance well from the FPU by controlling the direct memory access between the data memory and floating-point processing unit.

The FPUC works as the interface between the CPU and the FPU, the sequencer for the fundamental arithmetic calculation, and the address generator for vector operations. The FPUC is prepared with the LSI Logic's compacted gate array. The details of the FPUC will be described in the later section.

The floating-point operation system can work in three modes of operations; The first mode is the scalar operation, the second mode the microprogram operation, and the third mode is the vector operation.

(1) Scalar operation

The scalar operation is used for the scalar number calculation. In this mode, the CPU executes the move instructions between the register of the CPU and the high-speed data memory with the address bus signal including the instruction to the FPU (13bit) and the address of the data (19bit). The data to/from the register of the CPU is gated by the FPUC and the data transfer is actually done between the data memory and the registers of the FPU. By this mode, the CPU controls every step of floating-point operations of the FPU, and one data transfer and a pair of the subset of the pair of the multiplier and the ALU instructions can be executed by one move instruction.

(2) Microprogram operation

The microprogram operation mode is used for the elementary functions, random number generations, and the operation of 3x3 complex matrices. In this mode, the FPUC executes the operations according to the preset microprograms in the WCS.

(3) Vector operation

The vector operation mode is used for the

arithmetic operation of vectors and matrices. The CPU sets the initial addresses of the operands and the increments of the addresses into the registers in the FPUC, and sets the microcode into the WCS. The microcode is the vector operation parts in the object code of the user program. Then the CPU directs the FPUC to execute the microcode.

The high-speed floating-point operation system consists of ;

(1) a floating-point processing unit (FPU),
(2) a floating-point processing unit controller (FPUC),
(3) a look-up table (LUT) ROM,
(4) a writable control storage (WCS),
(5) and a high-speed floating-point data memory (DM).

The DM is the 32bit x 512Kwords memory of high-speed (35ns) 256Kbit CMOS static RAMs. The CPU accesses the floating point data memory with one wait state.

The look-up table (LUT) is the two 8bit x 512 words PROMs containing the initial data for the Newton-Raphson calculations of the inverse and the inverse of square root.

The WCS is the 32bit x 2Kwords high speed (25ns) BiCMOS static memory to store the procedure for the calculation of fundamental arithmetic, vector operations, and the other calculations necessary for the QCD calculations such as the 3x3 complex matrices multiplication. The CPU can access the WCS with no wait state.

4.4 FPUC (FLOATING-POINT PROCESSING UNIT CONTROLLER)

The FPUC was fabricated with CMOS gate array technology. The FPUC essentially consists of three parts; data transfer control part, data manipulation part and sequence control part.

The data transfer control part generates the address of the data written/read to/from the high-speed data memory and the enable signals of the output/input data registers of the FPUC and FPU. This part consists of sixty-four TRs (data transfer control registers) and IRs(data address increment registers) and an adder. TR is the 32bit register containing the 20bit data address and the enable signals of the input and output data registers of the FPUC and FPU. IR is the 16bit register holding the number which is added to the data address every time when the data address is read out. The 16bit increment is expressed by 2's complement format, and the sign extension makes the 20bit data for the input of the adder.

The data manipulation part consists of the two 32bit data registers and a data manipulating circuit. The functions are,

(1) Take bitwise exclusive or of D0 and D1,
(2) Set the exponent of D0 to 0x7F,
(3) Replace the exponent of D0 by bit7-0 of D1.

These functions are used for the random number generation, calculation of exponential and logarithmic functions.

The sequence control part executes the conditional or unconditional branch instructions, branch to and return from subroutine instruction, and detects the end of vector operation. During the vector operation, the data address is compared, and if it goes across the end address set in the FPUC the vector operation is terminated. As the FPUC has a register for the program counter stack, one level subroutine call is available. During the execution of main routine, the return from subroutine instruction is interpreted to the end of the FPUC routine and the control is returned to the CPU.

The high-speed floating-point operation system works under the three-stage pipeline operation sequence. Each stage of the pipeline is as follows;

(1st) Update program counter and fetch instruction from WCS.
(2nd) Read-out the information about data transfer from TR and IR. Update TR.
(3rd) Execute data transfer and FPU operation.

As the floating-point calculation is completed in the third stage only, the result just got in a clock period can be used for the calculation in the next clock period. So, this system executes the recurrence without an idle time.

4.5 INSTALLATION

One PU is installed on a six layer print board of 367mm x 400mm. The photo of a PU is shown in the Fig.3a. Sixteen PU's constituting 4x4 PU array are installed in a box. This box is called a module. A module includes a power unit and a repeater of the common bus. The photo of a module is shown in the Fig.3b. The QCDPAX system with 288 PU's consists of circularly connected six cabinets each of which contains three modules. As each cabinet can contain up to six modules, the maximum configuration of QCDPAX is 6x6 modules, i.e. 24x24 PU's. The photos of the cabinets (Fig.3c) and the three modules in a cabinet (Fig.3d) for the QCDPAX with 288 PU's are shown.

5. SOFTWARE

5.1 DEVELOPMENT OF THE PROGRAMS

The operating system of the host-computer is

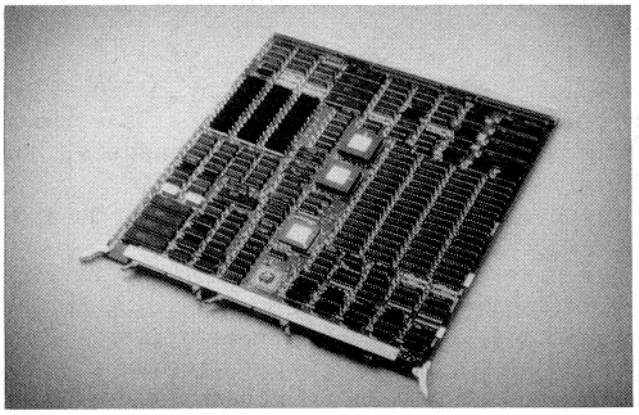


Fig.3a. A processing unit board.

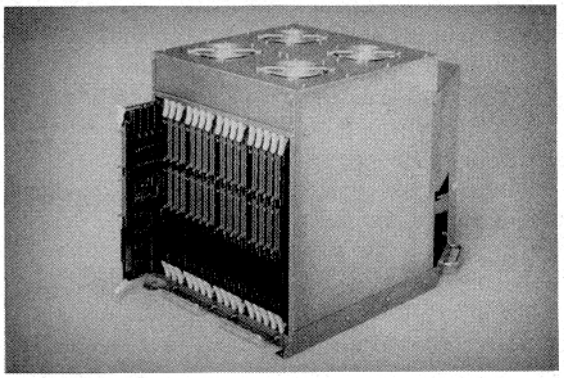


Fig.3b. A processing module with 4x4 PUs, a repeater of the common bus and a power unit.

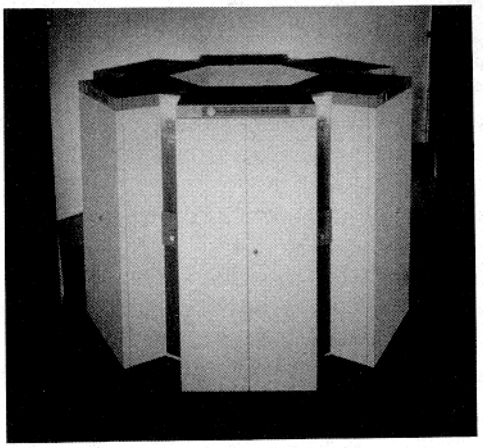


Fig.3c. QCDPAX with 288 processing units. Six cabinets form a cylinder.

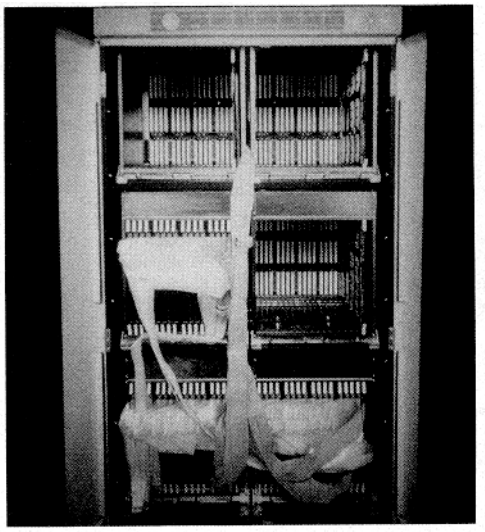


Fig.3d. Three processing modules in a cabinet.

UNIX, and supports multi-user and multi-task operation. But the PU array works in a batch style. Only one job is executed in the PU array, and the other job which intends to be loaded in the PU array is refused. The editing, compilation and linking of the user program are made on the host-computer in multi-user, multi-task style.

The user of QCDPAX prepares two programs. One is for the host-computer, and the other program is for the PU's. The program for the host-computer is described by the language C. A language psc is developed for the description of the PU program.

A compiler is prepared for coding the PU program. It will support the floating-point operations on the high-speed floating-point operation system. Tuning of the code in assembly language is effective in the case of vector operations. The assembly language qfa is specially designed, and assembler is also prepared.

The flow of the development of application software is illustrated in Fig.4.

5.2 THE DOMAIN AND THE TYPE OF VARIABLES

In order to set the variables to the appropriate memory, variables are declared with their domain name and type. The domain names are corresponding to the memories as shown in the Table 1.

Table 1. Domain name vs. memory.

domain name	corresponding memory
fast	data memory
slow	local memory
east	CM to the east PU
west	CM to the west PU
north	CM to the north PU
south	CM to the south PU

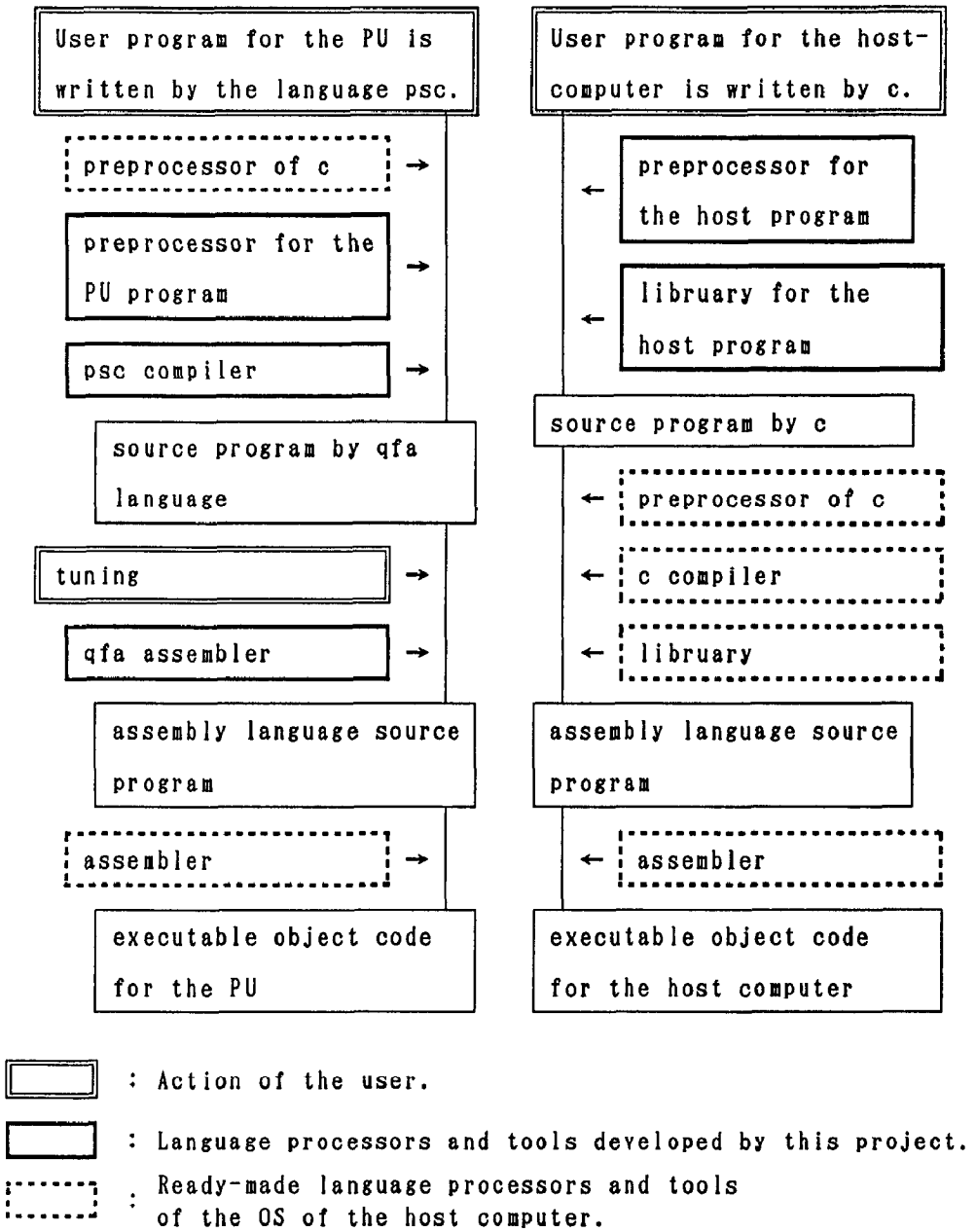


Fig.4. The process of developping an application program for the QCDPAX.

According to the variable declaration, the compiler for the PU program generates the appropriate object code and the preprocessor for the host program translates the variables to the member of structure mapped to corresponding memory.

As the type of variables, integer, float, and complex are available. The complex type is necessary for the lattice QCD calculations. Complex type variable is translated to a pair of float type variables by the preprocessors.

5.3 LANGUAGE psc AND ITS COMPILER

The psc is a c-like language for nodal program in a PU and can describe explicitly the vector processing in a PU, parallel operation by CPU and FPU. The vector processing is described by vfor-do statement. An example of vfor-do statement is following.

```
vfor(v=vstart;v<vmax;v+=vstep)
  sp[v*2+s]  = m[v] * la[v][p]
             + sp[v*2+s] ;
do {
  for(j=0;j<10;j+=1)
    l_ferry[j] = l[j] ;
}
```

In the psc program listed above, the block after vfor is executed by FPU and the block

after do is executed by CPU. The real or complex arrays sp, m and la are located in the domain fast i.e. the high-speed floating-point data memory. The do block describes the data transfer from slow to ferry. These two blocks are executed parallelly. In the vfor-do statement, either FPU or CPU waits until the other finishes the operation.

5.4 FLOATING-POINT OPERATION SYSTEM ASSEMBLY LANGUAGE qfa

Psc compiler generates the object code for the floating point operation system by the qfa which is the assembly language specially designed for this floating point operation system. The qfa has better readability than the assembly language of MC68020, and allows easier tuning of PU program to the user.
For example, the psc program

```
vfor(i=0; i<6; i+=1) a[i]=b[i]+c[i];
do;
```

is compiled to

```
tr00:  _c > c ;
tr01:  _b > b ;
tr02:   a > _a ;
tr03:   5 > e ;
ir61:   1 ;
ir59:  -1 ;
@lab0:
       tr01,ir61 ;     ;
       tr00,ir61 ;     ;
       tr03,ir59 & a=b+c ;     ;
       tr02,ir61 ; ja @lab0 ;
er:  #0 & fpc: @lab0 & start ;
       v_wait ;
```

Here b and c are the input registers, and a is the register for the result of addition in the FPU. Variables are expressed by the name of array in the source program with preceding _ .

The assembler of qfa translates the qfa source program to the instruction stream of the assembly language of the CPU for the CPU to write the instructions of the FPUC into the WCS.

6. EVALUATION OF THE PROTOTYPE OF QCDPAX

Fundamental performance has been measured on the prototype with four PU's. The clock rate of the floating-point system of the prototype is 80 ns, so the practical system which has the clock rate of 62.5ns will work about 20% faster.

6.1 PERFORMANCE OF A SINGLE PU

The execution time is measured for the elementary functions, vector operations, whetstone benchmark test, and linear equation solution by Gauss-Jordan scheme. The results are shown in the tables 2, 3, and 4. Table 2 lists the execution time of fundamental functions. The execution time by SUN-3/260 compiled with the options of optimization and 68881 co-processor.

Table 3. lists the execution speed and half performance vector length $n_{1/2}$[11] of vector operations. Tuning is added to the vector operation at the qfa level.

The execution times of Whetstone benchmark test and linear equation solution by Gauss-Jordan scheme are listed in the Table 4. The execution times by SUN-3/260 with the options of optimization and 68881 co-processor, and SUN-4/280 with the option of -O4 optimization are also listed.

Parallel computers often demonstrate the Mandelbrot set calculation. It took 22.2

Table 2. Execution time of fundamental functions. (unit: microsecond)

function	QCDPAX PU	SUN-3/260
inverse	0.839	14.0
1/sqrt	1.395	40.0
sin	2.565	52.0
cos	2.561	52.0
atan	3.080	56.0
exp	2.278	56.0
ln	1.959	40.0
rnd	0.373	-

Table 3. Performance of vector processing. (MFLOPS)

Evaluated operations(vector length=500)	scalar	Vector	($n_{1/2}$)
A(i)=B(i)+10.0	0.260	6.243	(66)
A(i)=B(i)+C(i)	0.178	4.164	(35)
A(i)=3.0*B(i)+6.0	0.402	12.484	(79)
A(i)=B(i)*C(i)+D(i)	0.270	6.246	(38)
A(i)=(A(i+1)+A(i-1))*0.5	0.320	12.485	(78)
S = S+X(i)*X(i)	0.388	22.160	(206)
A(i,j)=B(i,j)*C(i,j)+D(i,j) array size=256x256	0.206	4.443	(-)
if(A(i)>0) C(i)=A(i)+B(i) else C(i)=A(i)-B(i)	0.110	3.123	(39)
Inner product of 3x3 complex matrices	-	12.795	(1)

Table 4. Measured performance of a PU by benchmark test programs.

Benchmark test program	QCDPAX PU vector operation with tuning	SUN-3/260 Optimized	SUN-4/280 Optimized	Unit
Whetstone	9.133	0.847	3.000	MWIPS
Linear eq.	0.561	11.900	0.770	second

seconds to calculate this set on 1000x1000 points in the square cornered at (-2,-2i) and (2,2i) by one PU with the iteration limit of 100 times. In this case, which includes the vector operation with conditional branch, the performance of one PU was 10MFLOPS.

6.2 PERFORMANCE OF PARALLEL PROCESSING

The Mandelbrot set calculation and display by four PU's took 7.20 seconds, and the net calculation time was 5.56 seconds. The difference of these time is due to the display which takes 3.41 seconds to update 1000x1000 points. This time to display seems to be sufficiently small for the lattice QCD application.

The Mandelblot set calculation is not a good example for the parallel processing because it does not need the inter-PU communication. Parallel processing inherently includes overhead i.e. the time spent for the process which is not appear in the serial processing such as the synchronization and the inter-PU communication. The time for these processes must be sufficiently small. The times for the synchronization of all PU's and the broadcast 1024byte data from one PU to all PU's were measured and listed in Table 5. The times of the PAX-64J, the former model of PAX computer, are also listed for comparison. The speed of data transfer between the high-speed data memories of neighboring PU's was measured to be 2.7MB/s.

The efficiency of parallel processing E is defined by,

$$E = T_s/(T_p*P) \quad (3)$$

where T_s is the execution time by serial processing, T_p is the execution time by parallel processing, and P is the number of PU's. This equation is equivalent to;

$$E = (T_p-T_o)/T_p \quad (4)$$

if the algorithms for one PU in the parallel processing is same to that of serial processing. Where T_o is the overhead time.

The efficiency of the parallel processing described by the equation (4) in the prototype of QCDPAX was measured on the solutions of Laplace equation by relaxation method and linear equation by Gauss-Jordan method. The results are listed in Table 6. The algorithms used these sample parallel processing are consistent[12], so, the efficiency of parallel processing in Table 6 do not change with the increase of the number of PU's proportional to the problem size.

7. CONCLUSION

QCDPAX with 288 processing units has been constructed by the high-performance VLSI chips and the architecture based on the more than ten years history of the research of PAX computers. As the maximum speed of one PU is 32MFLOPS, the peak performance of the system with 288 PU's is 9.2GFLOPS. But the arithmetic logic unit and multiplier seldom works simultaneously, so the effective maximum speed of one PU is considered

Table 5. The execution time of synchronization and broadcast.

Operation	QCDPAX	PAX-64J	unit
Synchronization	2.5	18.0	microsecond
Broadcast(1024byte)	0.56	9.0	millisecond

Table 6. Total execution time, overhead time, and efficiency on sample programs.

Sample program	scalar processing			vector processing		
	total	overhead	efficiency	total	overhead	efficiency
Laplace equation	2.42s	0.0434s	0.98	0.220s	0.0434s	0.80
Linear equation	1.44s	0.0348s	0.98	0.214s	0.0348s	0.84

to be 16MFLOPS. At the practical lattice QCD simulation, the time for data transfer, the time for startup of the vector processing and the hyperplane method, and the time for net calculation are estimated to be almost same. Therefore the practical speed of 288 PU system is estimated to be 288*16/3=1.5GFLOPS. In the case of 480 PU system, the peak performance is 15GFLOPS and the practical speed is estimated to be 2.6GFLOPS.

This performance will satisfy the needs of lattice QCD simulation.

ACKNOWLEDGMENT

The QCDPAX project is conducted under the Grant-in-Aid for Specially Promoted Research of the Ministry of Education, Science and Culture of Japanese government (#62060001). It is a pleasure to acknowledge the strong support and encouragement by Prof. Kazuhiko Nishijima, Kyoto University, Prof. Akito Arima, University of Tokyo and Prof. Hideo Aiso, Keio University. The authors are grateful for the helps to Hiromu Komai, Jun Naito, Takeshi Yoshida, Yoshiyuki Yoshida, Shigeki Fujii, Youichi Hachikubo and other members of QCDPAX collaboration, as well as the staffs in Anritsu Corporation for their helps in the computer system development.

REFERENCES

[1] N.H.Christ, The Columbia Supercomputer Project: Physics Results Present Status and Future Plans, Lattice Gauge Theory (1986) Plenum, 159, pp.55-62.

[2] J.beetem, M.Denneau, and D.Weingarten, IEEE Proc. of the 12th Annual international Symposium on Computer Architecture (Washington, D.C., 1985); Experimental Parallel Computing Architecture, J.J. Dongarra, editor (Elsevier Science Publisher 1987) 255-298.

[3] The APE Collaboration: M.Albanese et al., Computer Physics Communications, 45 (1987) 345.

[4] T.Hoshino, T.Kawai,T.Shirakawa,J.Higashino, A.Yamaoka, H.Ito, T.Sato, and K.Sawada, PACS, A parallel microprocessor array for scientific calculations, ACM Trans. on Computer Systems, 1, 3 (1983) 195-221.

[5] T.Hoshino and T.Shirakawa, Load follow simulation of three-dimensional boiling water reactor core by PACS-32 parallel microprocessor system, Nuclear Technology, 56, 3 (1982) 465-477.

[6] T.Hoshino, T.Shirakawa, Y.Oyanagi, K.Takenouchi, and T.Kawai, Super Freedom Simulator PAX, in "VLSI engineering" (1984) edited by T.L. Kunii (Springer Verlag, Tokyo) 39-51

[7] T.Hoshino, T.Shirakawa, T.Kamimura, T.Kageyama, K.Takenouchi, H.Abe, S.Sekiguchi, Y.Oyanagi, and T.Kawai, Highly Parallel Processor Array "PAX" for Wide Scientific Applications, Proc. of the 1983 International Conferrence on Parallel Processing, (1983) 95-103.

[8] T.Hoshino, T.Kamimura, T.Iida, and T.Shirakawa, Proc. of the 1985 International Conferrence on Parallel Processing, (1985) 426-433.

[9] T.Hoshino, T.Shirakawa, and K.Tsuboi, Mesh-connected parallel computer PAX for scientific applications, Parallel Computing, 5, 3 (1987) 363-371.

[10] Y.Iwasaki, T.Hoshino, T.Shirakawa, Y.Oyanagi, and T. Kawai, QCDPAX: A Parallel Computer for Lattice QCD Simulation, Computer Physics Communications, 49 (1988) 449-455.

[11] Y.Oyanagi, Computer Physics Communications, 42 (1986) 333.

[12] R.W.Hockney and C.R.Jesshope, Parallel computers (1981) Adam Hilger, Bristol.

[13] J. J. Lambiotte,Jr. and R.Voigt, The Solution of Tridiagonal Linear Systems on the CDC STAR-100 Computer, ACM Trans. Mathematical Software, 1 (1975) 308-329.