lel Computer, Preprint n.737, Dipartimento di Fisica, Università di Roma *La Sapienza* (1990);

[6] A. Bartoloni, C. Battista, S. Cabasino, F. Marzano, P. Paolucci, J. Pech, F. Rapuano, R. Sarno, G. Todesco, M. Torelli, W. Tross, P. Vicini, R. Borgognoni, F. Del Prete, A. Lai, R. Tripiccione, N. Cabibbo, A. Fucci, Preprint n. 838, Dipartimento di Fisica, Università di Roma *La Sapienza* (1990);

[7] S. K. Ma, *Modern Theory of Critical Phenomena*, (Benjamin, New York, USA 1976);

[8] G. Parisi, *Statistical Field Theory* (Addison-Wesley, Redwood City, USA 1988);

[9] A. Bartoloni, C. Battista, S. Cabasino, N. Cabibbo, F. Del Prete, F. Marzano, P. S. Paolucci, R. Sarno, G. Salina, G. M. Todesco, M. Torelli, R. Tripiccione, W. Tross, P. Vicini, E. Zanetti, *MAD, a Floating-Point Unit for Massively-Parallel Processors*, Particle World 2, (1991) pp.65-73;

[10] The APE Collaboration, in preparation;

[11] F. Rapuano, talk given at the 1992 Amsterdam LAT92 Lattice Conference, to be published in Nucl. Phys. B (Proc. Suppl.).

[12] see for example G. Parisi, *A Short Introduction to Numerical Simulations of Lattice Gauge Theories*, in *Critical Phenomena, Random Systems, Gauge Theories*, edited by K. Osterwalder and R. Stora (North-Holland, Amsterdam, The Netherlands 1986); G. Parisi, *Principles of Numerical Simulations*, in *Fields, Strings and Critical Phenomena*, edited by E. Brézin and J. Zinn-Justin (North-Holland, Amsterdam, The Netherlands 1990);

[13] The APE Collaboration, in preparation;

[14] P. M. Farran et al., *The* 3081−*E Emulator*, in *Computing in High Energy Physics*, edited by L. O. Hertzberger and W. Hoogland (North-Holland, Amsterdam, The Netherlands 1986);

[15] R. Benzi, F. Massaioli and R. Tripiccione, to be published.

[16] N. H. Christ, *Status of the Columbia* 256-*Node Machine*, Nucl. Phys. B (Proc. Suppl.), 17 (1990) pp. 267-271, and references therein;

[17] D. Weingarten, *The Status of GF11*, Nucl. Phys. B (Proc. Suppl.), 17 (1990) pp. 272-275, and references therein;

[18] M. Fischler, *The ACPMAPS System*, preprint FERMILAB-TM-1780 (1992);

[19] Y. Iwasaki, K. Kanaya, T. Yoshie, T. Hoshino, T. Shirakawa, Y. Oyanagi, S. Ichii and T. Kawai, *QCDPAX: Present Status and First Physical Results*, Nucl. Phys. B (Proc. Suppl.), 20 (1991) pp. 141-144, and references therein;

[1] P. Bacilieri, S. Cabasino, F. Marzano, P. Paolucci, S. Petrarca, G. Salina, N. Cabibbo, C. Giovannella, E. Marinari, G. Parisi, F. Costantini, G. Fiorentini, S. Galeotti, D. Passuello, R. Tripiccione, A. Fucci, R. Petronzio, F. Rapuano, D. Pascoli, P. Rossi, E. Remiddi and R. Rusack, *The APE Project: a* 1 *Gflops Parallel Processor for Lattice Calculations*, in *Computing in High Energy Physics*, edited by L. O. Hertzberger and W. Hoogland (North-Holland, Amsterdam, The Netherlands 1986);
M. Albanese, P. Bacilieri, S. Cabasino, N. Cabibbo, F. Costantini, G. Fiorentini, F. Flore, L. Fonti, A. Fucci, M. P. Lombardo, S. Galeotti, P. Giacomelli, P. Marchesini, E. Marinari, F. Marzano, A. Miotto, P. Paolucci, G. Parisi, D. Pascoli, D. Passuello, S. Petrarca, F. Rapuano, E. Remiddi, R. Rusack, G. Salina and R. Tripiccione, *The APE Computer: an Array Processor Optimized for Lattice Gauge Theory Simulations*, Comp. Phys. Comm., 45 (1987), pp. 345-353; see also:
G. Parisi, F. Rapuano and E. Remiddi, *The APE Computer and First Physics Results*, in *Lattice Gauge Theories Using Parallel Processors*, edited by Li Xiaoyuan et al. (Gordon and Breach, London, U.K. 1987);

[2] For detailed reviews, see
N. H. Christ, *QCD Machines*, Nucl. Phys. B (Proc. Suppl.) 9 (1989) pp. 549-556;
R. Tripiccione, *Dedicated Computers for Lattice Gauge Theories*, Nucl. Phys. B (Proc. Suppl.) 17 (1990) pp. 137-145;
N. H. Christ *QCD Machines - Present and Future*, Nucl. Phys. B (Proc. Suppl.), 20 (1991) pp. 129-137;
D. Weingarten, *Parallel Q.C.D. Machines*, Nucl. Phys. B (Proc. Suppl.), 26 (1992) pp. 126-136;

[3] K. G. Wilson, *Confinement of Quarks*, Phys. Rev. D10 (1974) pp. 2445-2459;

[4] See for example:
S. Cabasino, F. Marzano, J. Pech, F. Rapuano, R. Sarno, W. Tross, N. Cabibbo, E. Marinari, P. Paolucci, G. Parisi, G. Salina, G. M. Todesco, M. P. Lombardo, R. Tripiccione and E. Remiddi, *The APE with a Small Jump*, Nucl. Phys. B (Proc. Suppl.), 17 (1990) pp. 218-222;
S. Cabasino, F. Marzano, J. Pech, F. Rapuano, R. Sarno, W. Tross, N. Cabibbo, A. L. Fernandez, E. Marinari, P. Paolucci, G. Parisi, G. Salina, A. Tarancon, G. M. Todesco, M. P. Lombardo, R. Tripiccione and E. Remiddi, *The APE with a Small Mass*, Nucl. Phys. B (Proc. Suppl.), 17 (1990) pp. 431-435;
S. Cabasino, F. Marzano, J. Pech, F. Rapuano, R. Sarno, G. M. Todesco, W. Tross, N. Cabibbo, M. Guagnelli, E. Marinari, P. Paolucci, G. Parisi, G. Salina, M. P. Lombardo, R. Tripiccione and E. Remiddi, *APE Quenched Spectrum*, Nucl. Phys. B (Proc. Suppl.), 20 (1991) pp. 399-405;

[5] N. Avico, P. Bacilieri, S. Cabasino, N. Cabibbo, L. A. Fernandez, G. Fiorentini, A. Lai, M. P. Lombardo, E. Marinari, F. Marzano, P. S. Paolucci, G. Parisi, J. Pech, F. Rapuano, E. Remiddi, R. Sarno, G. Salina, A. Tarancon, G. M. Todesco, M. Torelli, R. Tripiccione, W. Tross, *From APE to APE*-100*: From* 1 *to* 100 *Gflops in Lattice Gauge Theory Simulations*, Comp. Phys. Comm. 57 (1989) pp.285-289;
N. Avico, C. Battista, S. Cabasino, N. Cabibbo, F. Del Prete, A. Fucci, A. Lai, M. P. Lombardo, E. Marinari, F. Marzano, P. S. Paolucci, G. Parisi, J. Pech, F. Rapuano, R. Sarno, G. Salina, G. M. Todesco, M. Torelli, R. Tripiccione, W. Tross, P. Vicini, *A* 100 *Gflops Paral-*

chronous, and it has a very complex network (where APE has only a very simple communication pattern), which potentially allows it to deal with problems with a complex data structure (with a very high software effort). Till now the network has shown to be very useful for debugging, error detection and run-time reconfiguration purposes.

The ACPMAPS System is running at Fermilab, with a very advanced software system, Canopy (which is comparable to the level of the APE language, with its very effective optimizer we have described before). Different physics aspects of lattice Q.C.D. (mainly heavy quark physics) have been dealt with. The original ACPMAPS was 5 Gflops system. A board update (from a Weitek chip set to an Intel i860 based board) has pushed the peak speed to 50 Gflops, but the unchanged communication back-bone has started to show some inadeguacieses. An hand-recoding of QCD programs has shown that a sustained speed in excess of 20 Gflops can be reached on the updated machine.

QCDPAX [19] has been built at Tsukuba University, in Japan, counting on a long standing tradition of building computers. It has been running quenched QCD in the last two years. It is a MIMD computer, with 480 units (a few used as a backup). It has a toroidal $2d$ simple communication grid, a 14 Gflops peak speed, of which 2.5 are sustained for Q.C.D. codes. In contrast to the APE programming environment, the user has to write to explicit, separate codes, one for the host and one for the processing units.

Large scale simulations of Lattice Q.C.D. started indeed with early Cray super-computers, and today Cray YMP's are widely used. Programs are highly vectorized, and the efficiency is quite high. Commercial parallel machines have been, in the last year, becoming more and more competitive. Foe example Conjugate Gradient inversion programs on the largest available Thinking Machine CM-2 computers have been pushed up to 6 sustained Gflops. Very effective programs also run on the Intel Delta at Caltech, and efforts are already in progress to write effective codes for the new Thinking Machine CM-5 with its vector data path. Somehow the efforts on commercial computers have been complementary to the ones on dedicated computers. Commercial computers have been may be more useful in the process of developing new algorithms (but for exceptions, see for example the *smearing* techniques introduced in [4]), while dedicated engines have been crucial in pushing very large scale simulations, and getting reliable quantitative results.

Acknowledgements. It is a pleasure for us to thank Ettore Remiddi and Pedro Mato for useful discussions and suggestions.

REFERENCES

Let us now describe the principles of operation of the optimizer. Its structure is quite similar to that of the optimizer used in APE. In the first phase the optimizer tries to combine multiplications and additions into so-called **normal** operations for which the architecture of MAD is optimized. In a second phase (pipeline and device usage optimization) the optimizer rearranges and packs the code, attempting to find the position of each instruction that minimizes the total number of machine cycles. Two kinds of constraints are taken into account: the operands of an instruction must be calculated before being used, and two instructions cannot use the same hardware device at the same time. The cycle number at which each device of the computer (busses, registers, I/O ports, floating point adder and multiplier) is reserved by each instruction is defined by optimizer tables. Apart from these constraints, the optimizer is free to re-organize the code with the goal of filling the pipeline. The last phase consists in the allocation of the registers. Finally the executable code is produced.

5. Lattice Q.C.D. on dedicated and commercial computers.. The APE project has been developed in parallel with many other Q.C.D. machines. The relevance of the Q.C.D. lattice problem (and, as we have explained, more generally of lattice problems in Theoretical Computational Physics) has prompted many groups both to build optimized computers, and to use in an optimized way commercial computers.

APE runs a typical Q.C.D. matrix inversion conjugate gradient code with an a efficiency of order of .8 (order of 5 Gigaflops on the 6 Gigaflops machine). Because of the synchronous nature of the APE computer and of lattice QCD the efficiency will be basically the same on all size machines. Fluid dynamics problem also run on APE very effectively; on a 512×512 $2d$ grid the Lattice Boltzmann Equation is simulated at more than 2 sustained Gflops on the 6 Gflops machine [15]. Indeed, as we have already stressed, APE is not a *QCD engine*, but more a, quite general purpose, *lattice engine*.

A series of such computers has been developed at Columbia University [16], culminating with a 16 Gflops peak parallel computer, which runs some parts of Q.C.D. codes also with an efficiency very close to 90%. The machine has been running in 1992 mainly Q.C.D. thermodynamics, and behaves very reliably. TheColumbia computer is MIMD in nature, even if the practical use has always been very SIMD oriented; MIMD features have mainly been used for debugging and start-up purpose. A local, simple network allows synchronous communications. The part of the code which needs optimization has to be written in a low level assembler.

The GF-11 computer [17], built at IBM Yorktown, has a 10 Gflops peak speed, and sustains 7 Gflops on Q.C.D. problems (in 1991-1992 it has been running pure Q.C.D. spectroscopy). The computer is completely syn-

4.2. The compiling system. The compiler chain runs on the host computer. The main steps in the chain are the compiler (or the assembler) and the optimizer. The optimizer has some characteristics of generality which has allowed its porting from APE to APE-100 with only minor changes. In the following, we describe the organization of the compiler chain.

4.2.1. The *Apese* language. Here is an overview of the *Apese* language:

- there is a minimal subset of *Apese* that can be easily learned by every physics application oriented user. This subset is a structured language inspired by Fortran and C and is similar to the language used with the first generation APE machines;
- the language faithfully matches the APE-100 hardware architecture;

Even if the *Apese* language helps to write effective programs, an APE-100 user has to be well aware of some architectural characteristics. The programmer should consider APE-100 computers as a three dimensional mesh of nodes (from 8 up to 2048) with periodic boundary connections.

Data words on nodes are single precision 32 bits IEEE-754 standard floating point numbers. All the nodes execute the same code, typically and hopefully acting on different data. Parallel processing on APE-100 is limited to operations on floating point numbers. Every time the *Apese* programmer declares a floating point data structure memory will be reserved on all node distributed memories. The allocated memory will be placed on every node at the same local address, that will be associated to the name of data structure. Therefore each operation written in *Apese* language, acting on that name, will actually activate the same operation on every node of the mesh, choosing the data stored at the associated local address.

The programmer should also consider that integer arithmetic is performed by the Controller (which is in charge of the instruction flow and of addressing). As we have already discussed at length there are three type of conditioning that can modify the program execution: the first two IF and IF-ALL are managed by the Controller and may cause a true branch in the program flow, while the third one WHERE is locally managed by each node and may result in a temporary suspension of the effects of the program execution on that node, allowing a synchronous form of local program conditioning.

4.2.2. The optimization. During the first step in the compilation the compiler optimizes the register usage and the I/O to and from memory. The second step of the compilation chain produces code optimization. The optimizer reads the intermediate code produced by the compiler and generates the executable code.

FIG. 7. *3D connections of PB's and CB's. The cubes represent CB's, the rounded rectangles PB's. The link between PB's with* $(x, y, z) = (3, 0, 0)$ *and* $(3, 1, 0)$ *is highlighted in black.*

4. The software structure.

In the following we will discuss the operating system (OS), the compiler, the *Apese* language, the optimizer and the assembler language. A more detailed description can be found in [10].

4.1. The operating system.

As we have seen before the user controls APE-100 by means of the host computer, that provides a conventional file system and application development environment. The communication of APE-100 with the external world (host computer and mass storage devices) is asynchronous and is controlled by a network of Transputers.

To describe the functionality of the APE-100 OS it is useful to distinguish two different operating modes: user and system. The user writes programs in a high level or assembly language, from which code for the S-CPU and FPU's is generated. When an user program needs an asynchronous service (e.g. an I/O operation), it halts the S-CPU and switches to system mode under Transputer control.

The Transputer is able to gain control over the system also when abnormal conditions occur. Typical examples are the raising of an exception, or an operator request, or de-scheduling of an user program that exceeded some resource quota (for example the assigned maximum CPU time).

The code runs on the host computer, which contains the user interface called Host-resident APE Control Kernel. This provides the user with an interface to access APE-100 devices, issue I/O operations, load and run programs, monitor the machine status, etc.

FIG. 6. 256 *PB's in* 16 *crates in* 4 *racks.*

PB and is mounted on the opposite side of the backplane. Fig. 7 schematically represents the links between CB's and PB's for a fraction of the $3D$ lattice. A link in the Y direction is highlighted at the bottom left side of the figure. The connections in the X direction run on the crate backplane, those along the Y and Z directions run over bidirectional differential lines on twisted pair ribbon cables.

To understand the way in which the topology is implemented, note that each CB is assigned the (x, y, z) coordinates of the PB which precedes it in the direction of increasing X. Assume the range of the coordinates to be $0...N_x$, $0...N_y$, $0...N_z$. The Y and Z cables are laid according to the following rule: given a CB of coordinates (x, y, z) its *top* (as one can see in fig. 7) is linked to the *bottom* of the CB $(x-1, y, z+1)$, while its *back* is linked to the *front* of CB $(x-1, y+1, z)$, all coordinates being taken modulo N_x+1, N_y+1, N_z+1.

In conclusion, communication tasks are partitioned between PB's and CB's. This solution has several advantages with respect to the alternative (the PB's take care of all communication tasks, i.e. non CB's):

1. There are no cables directly plugged into the PB's, making them easily replaceable.
2. A smaller number of backplane pins is used for outboard connections and the complexity and pin count of the Commuter is also reduced.
3. The *hot* bipolar chips needed for buffering and level translation to/from differential lines are mounted far away from the *cool*, (mostly) CMOS chips on the PB's.

Fig. 5. *Connections between adjacent nodes.*

reducing the amount of cabling required for inter-crate links, while communication within a node is supported at full speed. In the case in which the communication field is known only at run time, the transfer time for a generic first neighbour displacement is a factor 4 larger than in the case of a local communication (on the same board, where the hardware bandwidth of one MAD plus memory is 50 Mbyte/sec) whose communication field known at compilation time. In this slower case the whole machine (MAD, memory and controller) has to remain idle, waiting for the transfer. This feature is implemented by stretching the clock for the needed number of cycles.

In a fully connected first neighbour $3D$ machine each PB would be provided with 6 links to neighbouring boards. However, in our design only one pair of links is active on each memory access: the left/right pair, as in fig. 5, the up/down pair or the front/back pair. For each given PB only two (not six) links are thus sufficient. We have left to a separate board, the Connection Board (CB) (which we will describe in the following) the further routing of these links to other boards.

In the full size, 100 Gflops machine, the 2048 nodes are arranged on a $8 \times 8 \times 32$ lattice. In this section it is convenient to view this pattern as a $4 \times 4 \times 16$ lattice of PB's, each holding a $2 \times 2 \times 2$ cube of nodes. The PB's are distributed among crates and racks as shown in fig. 6. Boards with the same value of Y and Z are in the same crate, and each crate holds four X-rows, displaced along the Y direction. A rack houses a full $X - Y$ plane of PB's.

There is one CB for every PB. The CB shares the same connector of the elementary conditions (i.e. $== 0$, > 0, $>= 0$, $=!0$, using C-language notations) which are made available to the IF circuitry. This is a 1-bit wide stack- based machine, which allows the evaluation of complex conditions by building on the Boolean operation AND, OR, NOT, to generate the *Local IF Status*. The stack allows nesting of up to eight conditional structures.

When the execution of a code section is conditioned (inside a WHERE block) and the *Local IF Status* is **false** all MAD operations are converted to NOP's (No-Operations), forbidding write operations to registers and memory. The AND of *Local IF Status* of all MAD's is delivered to the controller to obtain the global condition used by IF-ALL.

MAD is able to detect floating point exceptions (adder or multiplier overflow and LUT exceptions), non recoverable errors in the data bus and parity errors in the code bus. All the exceptions are maskable.

3.2.2. The communication network. Let us first discuss inter-node communication from the point of view of the application programmer, and then move on to hardware considerations. APE-100 is simple from the point of view of the programmer, since one can concentrate one's attention on the behaviour of a single node. The parallelism manifests itself through the existence of replicas of node data structures which are logically placed along the six possible directions in space. These replicas can be accessed by using six predefined constant displacements (corresponding to *Left, Right, Up, Down, Back and Front*). For instance, if $V(i)$ is an element of an array belonging to a given node, $V(i + Left)$ is the corresponding element on the closest left-side replica of the array. Inter-node communication can thus be controlled by an extension of the address field.

At the hardware level this model is implemented by splitting the address into two fields: a memory address, which is sent to the RAM chips, and a communication field, which is sent to the Commuter System. The value of the communication field does not need to be known at compilation time, and can be computed at run time.

The communication pattern is completely synchronous and only executes MAD to memory and memory to MAD operations at distance of one unit. If one moves to the *Right* a data from MAD to memory, then all MAD's move information to the memory at their *Right*, while all memories receive such information from the MAD at their left. All operations are synchronous, and no collisions are possible. There is no problem of contentions. Moving information at distance n must be done by repeating this elementary step n times.

As we show in fig. 5, for full communication speed during an inter-node transaction the PB should receive and transmit four data words. In practice, a $4:1$ time multiplexing in inter-node communications is implemented,

FIG. 4. *Simplified block diagram of MAD.*

check bits, which can transfer one data word every clock cycle. When MAD inputs a word, it uses the check bits to correct single-bit errors and detect double-bit (and some multiple bit) errors. In an output operations MAD generates a 7 bit modified Hamming code. Error correction is useful both against memory errors and inter-node transmission errors. Multiple errors are fatal and stop the machine. Single errors are not fatal, but on each occurrence MAD updates a counter which can be read when the synchronous backend is idle. The I/O bandwidth of the MAD is twice the bandwidth that can be actually used in APE-100, due to the memory access time and the absence of interleaving.

MAD contains Look-Up Tables (LUT's). LUT's are used to obtain approximated terms used in the computation of some frequently used functions like: $\frac{1}{x}$, $\log(x)$, $\frac{1}{\sqrt{(x)}}$, $\exp(x)$.

The arithmetic part of MAD consists of a multiplier and an adder hard wired for the **normal** operation: $D = (A \times (\pm B)) \pm C$. At each clock cycle one MAD can produce the results of one **normal** operation. Two of the 128 registers are permanently loaded with the values 0 and 1 so that simpler operations can be executed (without breaking the homogeneity of the code, which helps in producing well optimized codes):

- Simple sum: $D = (1 \times (\pm B)) \pm C$.
- Simple multiplication: $D = (A \times (\pm B)) \pm 0$.
- No-Operation: $0 = (0 \times (\pm 0)) \pm 0$.

The adder's result is used cycle by cycle to produce Boolean values for

FIG. 3. *Arrangement of nodes on a Processor Board.*

or remotely (i.e on neighbouring boards). Each of the chips composing the Commuter System dispatches 4 bits.

The cubic lattice topology is thus implemented through a distributed inter-connection system. Every PB also contains some support circuitry which allows connection of the PB's to the controller.

The PB's receive the MAD code from the Controller and deliver it to the 8 MAD's. While the MAD executes an instruction every clock cycle, the PB's are fed with a double instruction every two cycles. The instructions are unpacked locally and delivered every cycle. All the elements of the SIMD backend are driven by a single clock which synchronizes their activities cycle by cycle.

3.2.1. The MAD. Here we present a short description of MAD; more detailed information can be found in [9]. MAD (see fig. 4) is a 40 ns pipelined VLSI custom device controlled by a 48 bit wide control word supplied on a cycle by cycle basis.

The main components of the MAD are the register file, the floating point multiplier, the logic and arithmetic floating point unit, the look-up tables, the error detection and correction unit and a status register.

The register file contains 128 32-bit registers and supports 5 simultaneous accesses (3 read, 1 write and 1 read or write). Thus on each cycle it is possible to start one floating point addition, one floating point multiplication and one I/O operation.

Input/output MAD operations use a bidirectional port of 32 data and 7

There are 40 Mbyte of dynamic memory on each processing board. Each node uses 4 Mbyte to store data and 1 Mbyte to store the EDAC code used to correct single error, detect all double errors and some multiple errors.

The memory uses 4 Mbit memory chips organized as 1 Meg $\times$ 4 bit: each chip contains one bit of data belonging to four different nodes, so that after a complete failure of one chip (4 bits wrong) 4 nodes would detect a single error which would be corrected, and the machine would continue working properly.

The management of the remote data transfers are handled by the same logical circuitry as the address generation. From the programmer's point of view the first neighbour nodes can be seen as an extension of the local address space. The hardware takes care of respecting the appropriate timings.

The transfer of contiguous data from node memory to MAD proceeds at a rate of 1 word per S-CPU cycle. In the time interval we need to transfer a single word, four floating point operations may be executed on MAD. The data access to the memory of the first neighbour is 4 times slower.

3.1.3. The asynchronous interface. The LAI processor is a Transputer placed on the controller board. When the S-CPU is halted (this condition is always under LAI control) the LAI is able to assert all external busses of the controller and to access node memories. When the crate controlled by the LAI is in local mode, the LAI has complete control of it, including exception handling. Finally, the LAI monitors environmental conditions, like temperature or supply voltage, and issues alarms as appropriate.

The LAI handles exceptions produced by all nodes or by the controller. It manages the clock distributed to the synchronous sections, the global start/stop signal and it controls the run/halt condition of the S-CPU.

Exceptions are collected and delivered to the RAI which controls the CDU (Clock Distribution Unit). After an exception the RAI sends a global stop signal. If the machine is divided in crate wide segments, the exceptions are treated locally by the controller of the crate.

Each controller is synchronized via a common clock routed to all crates, and they all run the same program with the same data. The S-CPU are synchronized, under the control of the LAI, via a global start-stop signal generated in the CDU. If the machine is segmented (e.g into racks), there is a separate start-stop signal for each rack.

3.2. The nodes and the processing boards. As discussed above the nodes of APE-100 lie on a 3 dimensional simple cubic mesh. On a given PB there are 8 nodes, forming an elementary 2^3 cube. The PB also contains 10 Commuter chips (forming the Commuter System, see fig. 3), which provide mutual inter-connection between nodes either locally (i.e. on the same PB)

FIG. 2. *Controller block diagram.*

controller variables. The IF-ALL executes a logical AND of the conditional expressions (local IF status) on all the nodes which run the program and then a conditional branch set according to the result of the AND.

In the normal operation mode we expect all S-CPU to operate synchronously on the same data; in order to avoid disasters in this mode the memory content of all S-CPU must be the same. The whole LAI-RAI system is in this mode, acting under the same clock, the same instructions, the same data. They are, in this mode, physically duplicated but acting as identical entities.

We have decided to have an independent controller in each crate in order to avoid dispatching the instruction word, which is rather long, to all the different crates. In this way only one signal (the clock) keeps the computer synchronized, while all the other control streams are generated on all crates. As a byproduct of this choice the computer can be partioned dynamically at the single crate level. Controllers can act independently, and each crate can run a different user program. In this situation inter-crate communication is strictly forbidden. However the system software needed to operate in this mode has not yet been written.

3.1.2. Node distributed memory and address generation. Data Memory Addresses (DMA), identical for all nodes, are generated by the S-CPU or by the LAI and are converted into Memory and Commuter control signals. All the memory control circuitry is centralized in the controller board.

a Boolean function of global (in the above sense) variables.

The IF-ALL instruction is a global structure, executed by the S-CPU on the basis of the logical AND of local conditions evaluated by all nodes which take part in the computation. The WHERE structure is local: it causes a block of instructions to be enabled only in those nodes where a certain local condition (the Boolean result of computations done by the local MAD) is fulfilled. Synchronization is preserved by executing the conditioned code on all nodes but disabling write operations on the nodes which do not fulfil the logical condition. This construct allows an effective breaking of the SIMD sequence.

When multiple S-CPU are active in a given program, they will execute the same branch, be it an IF (they all have identical copies of the global variables), or an IF-ALL (they all share the conditions returned by the nodes which they control).

These basic structures can be used to implement complex ones, such as the typical Fortran-like DO loops, or new parallel ones, such as REPEAT-UNTIL-ALL or REPEAT-WHILE-ALL.

3. The hardware structure. We will discuss in the following the controller and the processing boards of the computer (with the floating point MAD chip and the communication network).

3.1. The controller. We show the controller sub-system in fig. 2. It includes the Synchronous CPU (S-CPU), program and data memories, the memory controller and a LAI for the communication with the external world. The whole sub-system is housed in a single board.

3.1.1. The Scalar CPU. The first implementation of the S-CPU [13] has been a synchronous integer processor, operating at a cycle time of 80 ns (similar to a processor developed at CERN as an improvement of the original $3081-E$[14]).

The S-CPU runs at one half the speed of the MAD: it executes one instruction for every four MAD floating point instructions (since MAD contains one adder and one multiplier). The S-CPU controls the program flow. The control word issued to the nodes can be regarded as an extension of the instruction word controlling the S-CPU itself: the same Program Memory Address (PMA) points at both control sections. In principle the nodes can be regarded as special purpose devices of just one processor: a controller branch directs all nodes. A single program instruction contains one S-CPU instruction and two MAD instructions (since, as we said, there is a difference in the S-CPU and the MAD clock speeds).

The S-CPU also executes global conditional structures: IF and IF-ALL. The IF instruction executes a test and branch on a logical condition on replication gives, among others, the advantage that APE-100 can be segmented into smaller independent units. The 2048-node version, for example, can be configured as four 512 node machines (or sixteen 128 node machines). Segmented configurations are supported by the Transputer network (which we will describe in the following), and are made possible by re-arranging subsections of the global mesh of processors in smaller blocks with periodic boundary connections. In this configuration communication among different machines is only supported by the Transputer network.

2.2. The asynchronous frontend interface. The asynchronous frontend interface can be seen as composed by two main elements: the host computer and the asynchronous interface. The host computer is the user interface: it runs the cross-compiler and the host resident part of the monitor/debugger. These services are available through local terminals or via a network. The host is connected to the synchronous backend by the asynchronous interface. Via this interface the host can load programs and data onto the synchronous backend, define its configuration, start and stop the execution of synchronous programs, and examine the result of the computation. The asynchronous interface is also responsible for the management of the APE-100 mass memories. It can save or load the content of the PB's memories to disks. The asynchronous interface is based on a network of Transputers which can be programmed to execute these tasks. The I/O facilities can be easily used from the user applications programs running on APE-100.

The asynchronous interface is distributed. A Transputer is the heart of the *Local Asynchronous Interface* (LAI), associated to each S-CPU and housed in the controller board. Another Transputer, the root Transputer, forms the *Root Asynchronous Interface* (RAI), connected to the host computer. In the processing board the Commuter chips have an asynchronous section accessible by the LAI. In the single controller configuration (i.e. up to 6 Gflops) there is no need for the RAI, as the LAI is directly connected to the host computer.

2.3. Data types and program flow. Our architecture naturally distinguishes two types of data: *global data* (on the S-CPU) and *local data* (on a node). In general, we regard S-CPU data as integer and local data as floating point (IEEE-754 standard). Local data reside in the node memories or on MAD registers, while global data reside in the S-CPU memory and registers.

Three kinds of conditional structures are implemented: IF, IF-ALL and WHERE. The IF structure is the classical non parallel Fortran-like instruction, which controls program flow according to a condition which is

Our choice has been to keep fully synchronous the largest possible part of the computer. That has made construction and testing possible in a very short time (and should make our computers very reliable). A microprocessor or a cache based RISC processors would not have satisfied such a standard. We wanted MAD to have 128 internal registers (and some commercial computers are now following us in this trend), which we judged to be a feature crucial for getting a highly optimized code. Our architecture which allows each processor to access a single register in one cycle is very flexible. This is different with the choice made, for example, for the CM-5 data path, where in the standard short instruction format mode registers are accessed in blocks of 8. We do not use vector registers, but we can sustain the full speed of our pipelining by accessing random registers (known at compilation time). Obviously the optimizer (which we will describe in the following) must be smart enough to deal with this feature. Our controller (the S-CPU) fits the MAD pace by construction, and the language is based on an optimizer which allows the attainment of peak performance from an high level dialect.

The heart of the APE-100 computing power is the floating point processor MAD [9], a custom VLSI device. MAD is a floating point 32 bit real arithmetic processor, and contains one adder and one multiplier. It has a peak floating point performance of 50 Mflops. The MAD architecture, which we will describe in the following, guarantees that a large fraction of this theoretical peak speed can be attained for an important set of numerical algorithms. MAD has built-in a large amount of support circuitry for error detection and correction. It contains specialized hardware for performing conditional write instructions and look up tables.

The communication is handled by a second set of custom VLSI devices, the Commuter System, which takes care of the node-to-node data transfer as well as of the communications with the asynchronous interface.

A processing node contains one MAD and its 4 Mbyte local memory. This is a reasonable memory size for Lattice Q.C.D., where a strongly increasing amount of computer time to solve the problem is required as a function of the lattice volume. In the largest machine we will produce we will have 8 Gbyte of memory, in which, as far as lattice Q.C.D. is concerned, we will be able to store a 64^4 lattice (for the standard quenched theory). Eight such nodes are assembled on a Processing Board (PB), together with a Commuter System. A 2048 node model contains 256 PB's (with a total of 8 Gbyte of memory) assembled in 16 crates (housed in 4 cabinets). The 128 node 6 Gflops version is built with 16 PB's which are housed in a single crate.

The S-CPU is housed in the Controller Board, and controls the nodes. The Controller is replicated, and there is one S-CPU in each crate. This

FIG. 1. *Schematic diagram of the controller and of the processing boards.*

fundamental variables are 3×3 complex matrices. Most of the computation is needed to multiply and add such matrices, or to multiply matrices and vectors. The problem is computationally intensive; each element of the matrix is needed many times in the course of a typical operation.

We regard QCD as a typical lattice problem, which can be solved in a very effective way by a *Lattice Engine.* in the same class there are many problems of high interest which are very similar to Lattice QCD as far as the computer demands and implementation is concerned. The only detail of our APE-100 architecture which has been strongly influenced by detailed features of the Q.C.D. lattice problem is the project requirement that memory-processor communication band-width should not be a bottle-neck in the design for this particular problem.

2. The APE-100 **architecture.** The architecture of APE-100, fig. 1, can be seen as composed of two layers: a synchronous backend and an asynchronous frontend interface. The software environment handles all the interactions, which are completely transparent to the user.

2.1. The synchronous backend. The synchronous backend is a Single Instruction Multiple Data (SIMD) $3D$ cubic mesh of nodes. Each node consists of a floating point unit (the *Multiplier and Adder Device*, MAD), a Communication and Control Unit System, composed by 10 identical chips (the *Commuter*) and a Memory Bank. The nodes are driven by a synchronous computer: the S-CPU [13]. All the elements of the backend are controlled by a 25 MHz clock.

TABLE 1
APE-100 at a glance - 1

Three dimensional topology.
Modular SIMD architecture (from 8 to 2048 nodes).
4 Mbyte, 128 registers and 50 Mflops per node.
Hardware support for local (non-SIMD) conditional structures.
Centralized address and integer computations.
25 MHz master clock.
8 nodes per Processing Board.
1 to 16 crates, each with 16 Processing Boards and 1 Controller.

TABLE 2
APE-100 at a glance - 2

	1 PB	1 Crate	Max Conf
Floating Point	400 Mflops	6.4 Gflops	100 Gflops
Data Memory Size	32 Mbyte	512 Mbyte	8 Gbyte
RAM to reg rate	400 Mbyte/sec	6.4 Gbyte/sec	100 Gbyte/sec

one could also build a 32768 node machine on a $8 \times 64 \times 64$ lattice, with a peak speed of 1.6 Tflops, but at present the construction of such a machine is not planned. The viability of this project has been especially due to the large use of custom VLSI components[9] and to the large effort devoted to the software development[10].

Let us just very briefly summarize here the present situation of the APE machines (see [11]). The old APE series (up to 1 Gflops) has been running for more than 4 years, helped to produce order 20 physics papers, and is now obsolete. One 6 Gflops machine has been running for the last 9 months (as of October 1992), dealing mainly with Q.C.D. and with fluid-dynamics simulations. Three more equivalent machines will be ready before the end of 1992, together with the first 6 Gflops machine built on smaller boards. This new model is identical to the old one, but more compact. We expect that the 100 Gflops machine will be running before the summer of 1993 (together with a smaller, 25 Gflops machine).

1.1. Lattice QCD. From a general point of view doing numerical simulations for lattice Q.C.D. [3, 12, 4] corresponds to solving a parabolic (and sometimes hyperbolic) differential equation using finite difference methods. In most of the cases the derivatives are approximated by first and second neighbour differences. One of the peculiarity of lattice Q.C.D. is that the

ABSTRACT

We describe APE-100, a SIMD, modular parallel processor architecture for large scale scientific computations. The largest configuration that will be implemented in the present design will deliver a peak speed of 100 Gflops. This performance is, for instance, required for high precision computations in Quantum Chromo Dynamics, for which APE-100 is very well suited.

Keywords: Parallelism, architectures, floating point, VLSI.

1. Overview. In the years 1985–1987, the APE collaboration has been involved in a major effort to design and build a parallel computer in the 1 Gflops range. APE [1] has been one among several projects [2] that have built floating point engines mainly tailored to the requirements of numerical simulations of Lattice Gauge Theories (LGT) and especially of Lattice Quantum Chromo Dynamics [3] (QCD), the gauge theory which, in the continuum limit, is supposed to describe the strong interactions between elementary particles. Three APE units, featuring floating point performances between 256 Mflops and 1 Gflops, have been completed and have been heavily used for LGT simulations. A large variety of physics results have been obtained on the APE computer. We have been studying, for example, the pure gauge and the hadronic mass spectrum in quenched QCD, and the deconfinement phase transition of quarks lattice QCD [4].

We will describe here the architecture of the next step, APE-100, a design that, by including all of the positive features of APE, has been able to push the peak speed to 100 Gflops [5]. The details of the hardware and software implementation of this architecture can be found elsewhere[6]. In Tables 1 and 2 we give the basic technical information about the APE-100 computers.

We recall that although APE was used mainly for Lattice Gauge Theory simulations, both APE and APE-100 are general purpose SIMD computers. The architecture of APE-100 is more flexible than that of APE and we plan to use APE-100 for many physical applications beyond LGT (e.g. fluid dynamics).

The SIMD architecture of APE-100 is based on a three dimensional cubic mesh of nodes with periodic boundary conditions, each node being connected to its 6 neighbours. APE-100 has a modular architecture. The building block is a $2 \times 2 \times 2$ cube, while a 2048-node APE-100 can be seen as a $8 \times 8 \times 32$ lattice. The 6 Gflops version is implemented with a simple array of 16 cubes. Such connection grids are well suited for the simulation of homogeneous physical systems (including, of course, Lattice Gauge Theories and Lattice QCD, and Statistical Lattice Systems [7], [8]). Discussing various aspects of the APE-100 design we will stress here the new ideas and implementations which allow the feasibility of a 2048-node machine, with a floating point performance of 100 Gflops peak speed. In principle the design is such that

THE APE-100 COMPUTER: (I) THE ARCHITECTURE

CLAUDIA BATTISTA, SIMONE CABASINO, FRANCESCO MARZANO,
PIER S. PAOLUCCI, JARDA PECH[*] FEDERICO RAPUANO,
RENATA SARNO, GIAN MARCO TODESCO,
MARIO TORELLI, WALTER TROSS and PIERO VICINI
Infn, Sezione di Roma
and Dipartimento di Fisica
Università di Roma La Sapienza
P. Aldo Moro, 00187 Roma (Italy)

NICOLA CABIBBO, ENZO MARINARI[†]
GIORGIO PARISI and GAETANO SALINA
Infn, Sezione di Roma Tor Vergata
and Dipartimento di Fisica
Università di Roma Tor Vergata
V. della Ricerca Scientifica, 00173 Roma (Italy)

FILIPPO DEL PRETE, ADRIANO LAI,
MARIA PAOLA LOMBARDO[‡]and RAFFAELE TRIPICCIONE
Infn Sezione di Pisa
56100 Pisa (Italy)

ADOLFO FUCCI
Cern
Geneva (Switzerland)

[*] on leave from Institute of Physics, Czechoslovak Academy of Sciences, 18040 Prague 8, Czechoslovakia

[†] and NPAC and Physics Department, Syracuse University, Syracuse, N.Y. 13244, USA

[‡] and Physics Department, University of Ilinois at Urbana-Champaign, Urbana, IL 61801, USA