Computer Physics Communications 45 (1987) 345–353
North-Holland, Amsterdam

THE APE COMPUTER: AN ARRAY PROCESSOR OPTIMIZED FOR LATTICE GAUGE THEORY SIMULATIONS

M. ALBANESE [d], P. BACILIERI [a], S. CABASINO [b], N. CABIBBO [c], F. COSTANTINI [d], G. FIORENTINI [d], F. FLORE [d], L. FONTI [a], A. FUCCI [e], M.P. LOMBARDO [d], S. GALEOTTI [d], P. GIACOMELLI [h], P. MARCHESINI [c], E. MARINARI [c], F. MARZANO [b], A. MIOTTO [f], P. PAOLUCCI [b], G. PARISI [c], D. PASCOLI [f], D. PASSUELLO [d], S. PETRARCA [b], F. RAPUANO [b], E. REMIDDI [a,g], R. RUSACK [h], G. SALINA [b] and R. TRIPICCIONE [d]

[a] *INFN-CNAF, Bologna, Italy*
[b] *Dipartimento di Fisica, I Universita' di Roma "La Sapienza" and INFN-Sez. di Roma, Italy*
[c] *Dipartimento di Fisica, II Universita' di Roma "Tor Vergata" and INFN-Sez. di Roma, Italy*
[d] *Dipartimento di Fisica, Universita' di Pisa and INFN-Sez. di Pisa, Italy*
[e] *CERN, Geneva, Switzerland*
[f] *Dipartimento di Fisica, Universita' di Padova and INFN-Sez. di Padova, Italy*
[g] *Dipartimento di Fisica, Universita' di Bologna and INFN-Sez. di Bologna, Italy*
[h] *The Rockefeller University, New York, USA*

The APE computer is a high performance processor designed to provide massive computational power for intrinsically parallel and homogeneous applications. APE is a linear array of processing elements and memory boards that execute in parallel in SIMD mode under the control of a CERN/SLAC 3081/E. Processing elements and memory boards are connected by a 'circular' switchnet. The hardware and software architecture of APE, as well as its implementation are discussed in this paper. Some physics results obtained in the simulation of lattice gauge theories are also presented.

1. Introduction

APE (Array Processor with Emulator) is a high performance computer designed to provide massive number-crunching capabilities for applications that are intrinsically parallel and homogeneous. So far, APE has been used for lattice gauge theory (LGT) simulations.

The full scale machine consists of a linear array of 16 cells, each consisting of a floating point processor and a memory-board. Floating point processors and memories are connected through a 'circular' switchnet. The number of cells is in principle arbitrary and can eventually be enlarged. The array runs in SIMD (Single Instruction Multiple Data) mode under the control of a CERN/SLAC 3081/E (the Controller), integrated in a general purpose host environment (presently a VAX/VMS system).

The theoretical speed of the machine is 1 Gflops while the total memory size is 256 Mbytes.

As an example of performance, a prototype consisting of 4 cells updates one link in an SU(3) pure gauge theory in 40 μs, running a program written in a high-level specially developed language. The efficiency for such a program, whose source code is larger than 1500 lines, is 70% of the theoretical speed. This figure can be compared with 35 μs obtained on a CRAY-XMP1 code in which the most time consuming routines were written in assembly language.

APE can efficiently perform many primitive computations such as matrix manipulation, fast Fourier transform, multidimensional convolution and, in general, algorithms that can be parallelized while requiring extensive communication among the processing elements.

While achieving a high throughput, typical of special-purpose processors, APE has a high degree of programmability. A high level FORTRAN-like language has been developed to take full advantage of the features of the machine. Although

0010-4655/87/$03.50

the user must keep in mind the parallel structure of the machine while writing application programs, no deep understanding of the structure of APE is required to achieve good efficiency. An optimizing step is in fact included in the APE compiler.

Two prototypes consisting of 4 cells each have been fully developed and are operational since september 1986. They have been used for SU(3) LGT simulations. Production of the full scale machine is underway (partially contracted to industrial firms). A full scale operational prototype is expected for spring 1987.

This paper describes the hardware and software architecture of APE and its implementation. We first give an overview of the machine and then give a more detailed description of the hardware structure of APE. Software aspects are then considered while some physics results and concluding remarks are presented in the last section.

2. Overview

APE, as shown in fig. 1, is a linear array of SIMD processors. A linear array is easy to implement, and can be expanded to larger size in a straightforward way. Furthermore a more complex structure (e.g. a planar array) can be efficiently simulated on a linear array, the reverse being not always true.

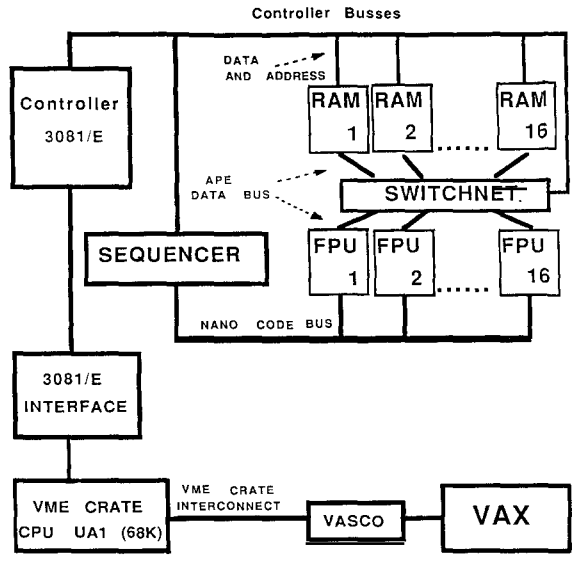

Fig. 1. Block diagram of the APE computer.

A SIMD architecture is simple from the point of view of implementation, since it requires only one sequencing unit, which controls all the processing elements via a broadcast micro-code. It also has conceptual simplicity, in that all problems of synchronization between processors are avoided.

The range of problems that can be solved on a SIMD machine clearly depends on the communication capabilities of the switchnet. As far as LGT simulation is concerned communication between first neighbor processors is adequate. This solution would however over-specialize our SIMD machine, preventing its use in even moderately non-local problems.

These considerations lead us to consider a 'circular' switchnet, capable of connecting in a 'rigid' way each FPU to each memory. Specifically the switchnet can establish a bi-directional data-flow between FPU(k) and memory($k+l$), $0 < k,l < 15$, periodic boundary conditions being used to wrap around the edge of the array.

This structure is adequate to handle non-local problems, provided that they can be solved by parallel algorithms. Still it is relatively simple and requires a limited number of control signals. Its simplicity can be exploited to attain a speed sufficient to sustain the peak performance of the processing elements. APE is supported by a specially developed software. The APE software is essentially a two level package, designed to adress two main issues. At the user level, a FORTRAN-like language naturally reflects the parallel structure of the machine. At a level closer to the hardware architecture of the machine, maximization of the performance of the pipelines of the machine is the key issue. This fact has favoured the decision to build a fully microcoded machine, completely avoiding any assembly-language level of programming. From the point of view of the high level user, APE is controlled by a host computer (currently a VAX). The Host resident APE Kernel Software (Hack) provides a VMS environment for the high level user. Hack consists of the APE compiler, the symbolic I/O manager, the

Debugger, the Program Loader and the Backup software.

3. Hardware

The APE computer has 4 elements: an array of 16 'cells', a switchnet, a sequencer and a controller running synchronously with a clock cycle of 120 ns.

Each cell comprises one Floating Point Unit (FPU) and one memory board. The switchnet provides a data-path between memories and FPU's. The connection is usually between the FPU and memory belonging to the same cell, but the data-path can be re-directed under program control to bi-directionally connect FPU's and memories of different cells. A sequencer broadcasts the same microcode (labelled nano-code in APE jargon) to all the FPU's. Nano-code sequences are stored in the (writable) sequencer memory and are broadcast to the FPU's under control of the sequencer itself.

The controller executes the integer-arithmetic and logic sections of the application program that are not mapped on the FPU's. It also generates appropriately sinchronized addresses and controls for the memory banks and the switchnet. Finally it controls the logic flow of the FPU program supplying branch adresses to the sequencer. The controller is connected to a VME bus via dedicated boards. The VME QBUS/UNIBUS connec-

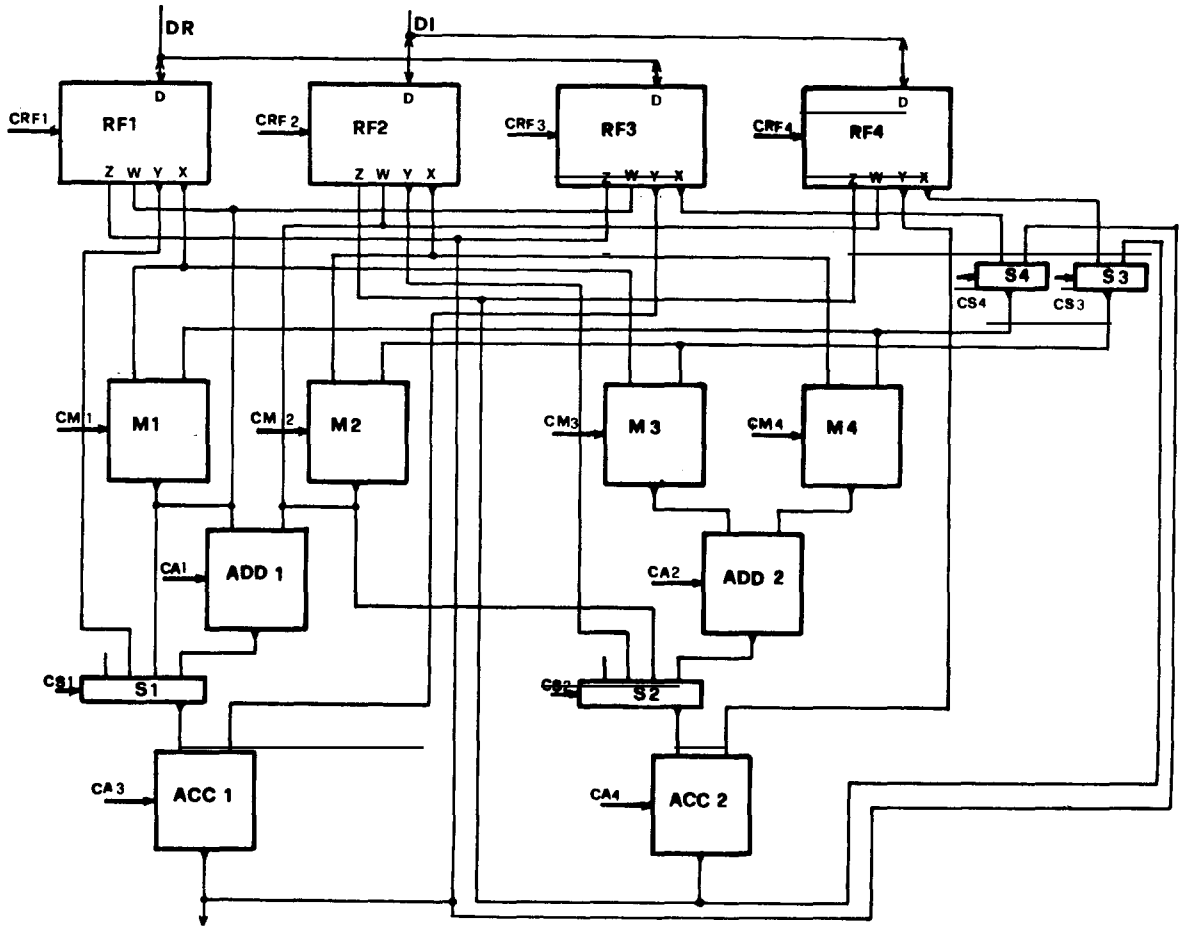

Fig. 2. Block diagram of the data-path of the Floating Point Unit of APE.

tion is finally accomplished through a dedicated interface. These hardware items are now described in more detail.

3.1. FPU

The Floating Point Unit is the processing element of the elementary APE cell. It has been designed to efficiently operate on complex floating point numbers with 32 bit accuracy. The floating point format conforms to the IEEE standard. A block diagram is presented in fig. 2. Communication to/from the memory is accomplished through two complex register files (each containing 32 registers). These are implemented with 4 Weitek WTL1066 register file chips, connected to the memory port through their bi-directional ports.

The FPU is optimized to perform the operation

$$A = B*C + D. \quad (1)$$

A, B, C, D being complex numbers. The FPU uses four floating point multipliers and four floating point ALU's, respectively Weitek WTL1032 and WTL1033 chips. They can start a new operation every clock cycle in pipeline mode, with a latency of 5 cycles. A suitable configuration of these devices is capable of obtaining a new result of operation (1) every clock cycle, after pipeline startup. Such a configuration is shown in fig. 3. The process of computing eq. (1) can be logically split in three steps. All products of the four real numbers in $B = (b_1, b_2)$ and $C = (c_1, c_2)$ are evaluated in step 1. The complex number $B*C = (b_1c_1 - b_2c_2, b_1c_2 + b_2c_1)$ is evaluated in step 2 by appropriately adding/subtracting the result of step 1. Finally $B*C$ and D are accumulated onto A in step 3 and the result is saved onto any of the two register files for later use or transmission to the memory. The latency of the full operation is 18 cycle.

The data-path can be re-arranged for greater flexibility on a cycle by cycle basis. For instance pure real arithmetic can be performed efficiently using the real and imaginary parts of each data word as two independent real numbers.

Special hardware is used to implement IF...THEN...ELSE structures supported by the APE language. On our SIMD machine IF structures can be implemented at a local and global level. At the local level writing of data onto memory is performed or inhibited in each cell according the value of the condition code generated on the cell itself by the (local) IF instruction. The local IF structure cannot alter the logical flow of the program. The latter can be modified by the global IF instruction also implemented in hardware. In this mode, the logical AND of the local condition codes is passed to the controller which will act consequently. In both cases four levels of nesting are hardwired.

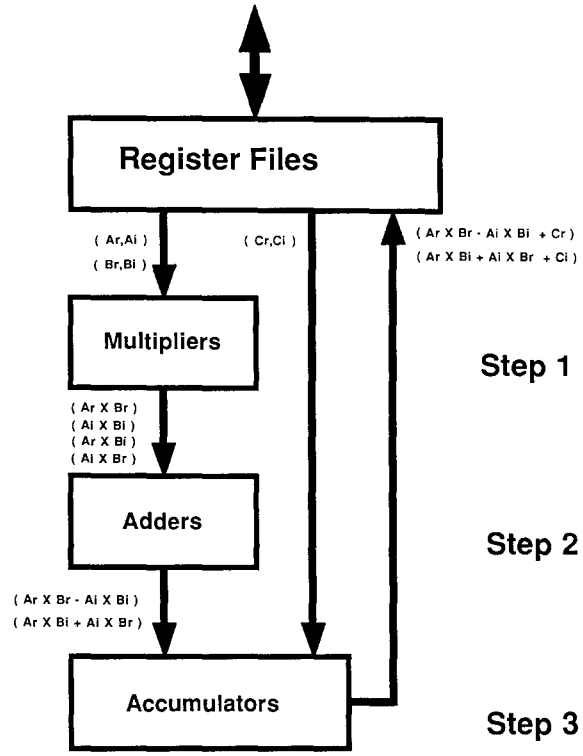

Fig. 3. Simplified diagram of the data-path of the Floating Point Unit of APE, showing the pipeline structure optimizing the evaluation of complex scalar products.

Heavily used special functions like exp and log are evaluated in the FPU's using a first approximation provided by hardwired logic. This is also true for the sqrt and inverse functions, with the help of an 8 bit look-up table built in the register files.

3.2. Memory

Typical LGT applications require a large data base (our long standing goal is a memory size of 1 Gbytes). This requirement dictates the use of dynamical memory. On the other hand there is a requirement of a large memory band-width, sufficient to keep the FPU's busy. As an example, the calculation of Tr(A*B*C*D), A B C and D being SU(3) matrices (a typical kernel of LGT codes) requires about 200 floating point operations on 24 complex numbers (or 200 bytes). Hence a band-width of 8 bytes per cycle is required. The conflicting requirements of large size and high band-width are met by organizing the memory in 8 interleaved banks, accessed in sequence. Memory access time is one word every three clock cycles for sparse access.

Higher speed is achieved in FAST mode. In this mode a block of memory up to 32 kbytes can be transferred at a rate of one word (8 bytes) per cycle after a start-up of 3 cycles. The 64 bit wide word is organized as 2 banks of 32 bits each. Seven check bits are used for each half-word to provide single error correction and double error detection, adding to the reliability.

8 Mbyte boards have been built using standard 256 kbit DRAM chips. The two working APE prototypes use such boards. The final version will use SIP packaging of 256 kbit chips to squeeze 16 Mbytes of memory into one board, giving a total memory of 256 Mbytes. This figure will be expanded to 1 Gbytes when denser chips are available at reasonable prices.

Each memory board can be accessed by the controller and the FPU on independent busses, addressing and control being provided by the controller in both cases. A common bus is shared by all memories for communication with the controller, while separate busses connect each memory to an FPU via the switchnet. Either a data transfer between one of the memories and the controller is active or a parallel transfer between each memory and the connected FPU is in progress at each clock cycle.

3.3. Switchnet

The switchnet provides a re-configurable data-path connecting FPU's and memories.
As already mentioned, the switchnet can connect FPU(k) and memory[mod($k+j$, 16)] $0 < k, j < 16$. The offset j is specified by an appropriate number of bits in the memory address.

The switchnet is a separate board housed between the crate containing the memories and the FPU crate. Flat cables ensure electrical continuity. Physically each data-path is 32 bits wide. The transfer rate is 1 64-bit word per clock cycle, multiplexed as 2 half words of 32 bits each.

Two prototypes connecting the FPU's and memories of 4 cells have been built using wire-wrap technology. Extensive use of programmable logic (PALs) has been made. The switchnet is transparent, in that the transfer time of one data word through the switchnet is shorter than 1 clock cycle.

The full scale version will follow a pipelined scheme. The transfer of 1 data word will take one full clock cycle. This brings the effective pipeline latency of the memory from 3 to 4 cycles, negligibly affecting the overall performance of the system. This choice has been made to ensure appropriate reliability to a device expected to deliver a band-width of 1 Gbyte/s and to have sufficient design margins to allow experimentation of more complex switchnet architecture in the future.

Using off-the-shelf components for the switchnet would result in an oversized board of high power consumption. To overcome these problems, we have designed a semicustom chip based on 2 μm CMOS gate array technology. This device, built by Plessey Ltd., connects one bit in the data-path across the sixteen FPU's and memories. Pipeline flip-flops are contained inside the chip. The switchnet requires 32 such chips, packaged in standard 40 pin DIP's. A prototype of the board is being tested at present.

3.4. Sequencer

The APE computer has one dedicated sequencer board, broadcasting the same nano-code to all the FPU's on a dedicated bus. The nano-code, controlling the cycle by cycle status of the machine, is 64 bits wide.

APE is a microcoded machine, completely bypassing any assembly language level of control. As a consequence, the size of the control store is large, nano-code sequences being usually very long (typically of order of 100–500 machine cycles). At present, the size of the writable control store of the APE sequencer is 16 Kwords of 96 bits each, to be expanded to 64 kwords in the near future. A 64 bit pattern is broadcasted to the FPU's, while 32 bits are used within the sequencer to control the flow of the nano-code stream. Sequencing is accomplished using a standard architecture based on the AMD-2909 μsequencer chips. Four AMD2909 are used to provide 16 bit addressing. The writable control store uses static CMOS RAM chips (16 k × 1 bit, to be replaced by 64 k × 1 bit) with access time of 45 ns. Parity bits are also stored (on a byte basis) and are checked at every clock cycle.

3.5. The controller

The controller performs all integer arithmetic and logical operations that are not mapped onto the FPU array. It also controls the operation of the FPU's through the sequencer and generates addresses for the memories and controls for the switchnet.

The controller is based on a CERN/SLAC 3081/E processor, described in details elsewhere [1]. This processor is well suited as the APE controller, since it is a synchronous machine with a clock period of 120 ns, well matching the FPU clock. It also has integer processing power adequate to fulfill the addressing requirements of the memory array. Finally, the 3081/E is interfaced to the VME bus via dedicated boards.

3.6. Host interface hardware

The host system of the APE machine is a VAX/VMS computer. The connection is done in two steps. First, the 3081/E is interfaced to the VME bus via dedicated boards. Programs running on a VME CPU (at present a Data Sud CPUA-1) control the 3081/E at the hardware level. Communication between VME and the VAX is accomplished via a specially developed VME-QBUS/UNIBUS controller, having a peak transfer rate of 500 kbytes/s. The main advantage of this configuration is that of keeping the user level and the hardware level well separated, allowing a relatively easy migration to different host systems. In fact, we are considering at present the possibility of using a VME system running UNIX as the host of APE.

The current APE implementation uses conservative design principles. TTL-compatible parts are used throughout and no custom-made components are used. Custom components have been developed and will be used in the switchnet for the full scale machine however. Fig. 4 is a picture of one 4 cells APE prototype. A single 19″ rack houses the 3081/E the memory banks and the FPU array. The two first prototypes use wire wrap technology. The full scale prototype is being built using PC cards.

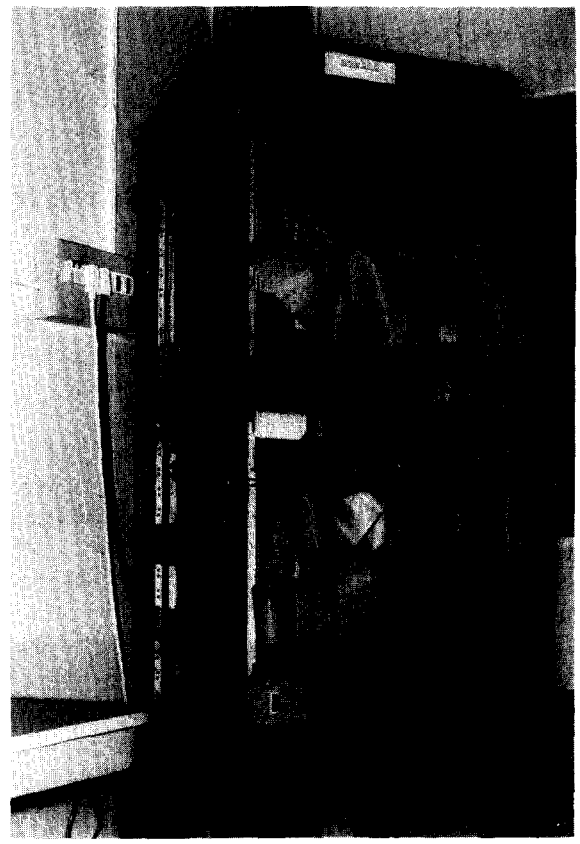

Fig. 4. Picture of the APE prototype #2.

4. Software

High level users can fully control APE capabilities while logged onto the host computer.

A typical APE working session can be described as follows:
The user logs onto the host computer and edits an APE language source program. He then invokes the APE language compiler to obtain an APE executable file and subsequently loads the executable code on APE, runs and debugs it, examines computed results and backups them using Hack: the Host resident APE Control Kernel.

4.1. The APE language and its compiler

APE language is a structured language quite similar to Fortran, even though it faithfully reflects the APE SIMD architecture.
The flow control is managed by language instructions directed to the controller: a branch is generated if data on the controller or cells satisfy the programmed condition.
Control statements are divided into two classes:
The first group consists of:
- if(condition)then... else... endif,
- for... endfor,
- repeat... until(condition),
- while(condition) endwhile.

These statements are used when the tested data reside on the controller memory: typically an address or loop counter.
The second group refers to parallel processing:
- where(condition)... endwhere,
- ifall(condition)then... elseall... endifall,
- repeatall... untilall(condition),
- whileall(condition)... endwhile.

The "where(condition)... endwhere" executes the nested instruction block only on the FPUs where data satisfy the required condition.

Using control statements such as "ifall(condition)then... endifall", the condition must be referred to data residing on FPUs. The global condition will be statisied only if it is true on all cells.

Data storage is allocated by explicit declaration of "integer" (32 bits), "real" (32 bits) or "complex" (32 + 32 bits) variables and multi-dimensional arrays.

However the instruction set allows closer control on data storage. When scalar variables are declared, data storage may be forced by specifiers:
- register: data will reside on registers (FPUs or controller).
- static: data will reside on memories (cells or controller).

Arrays of FPU registers may be declared: they are named "fast" arrays because a DMA is performed when loading or storing them. "fast in" and "fast out" statements permit data transfer between static and "fast" arrays. When needed the "allocate... endallocate" statements actually allocate controller registers for local variables.

The compiler automatically optimizes sequencer code by rearranging it. A good APE programming experience will improve code efficiency, relieving part of the burden of the optimizing step of the compiler. An expert APE programmer will minimize controller branches during inner loops to reduce pipeline breaking. He/she will obtain maximum performance using, as much as possible, controller and FPUs register variables and "fast" arrays when vector processing is required. Source optimization is facilitated by optimization symbolic data produced by the compiler. These tables allow easy identification of time critical source segments.

A preprocessor supporting macro expansion, token definition and source text inclusion is available. Its performance is comparable to that of a C language-preprocessor, while APE oriented improvements are implemented.

The compiler saves the symbols tables for the symbolic Hack debugger. Time synchronization between controller and sequencer micro-codes is performed by a time-linker.

4.2. The system software

The structure of the system software of APE present two levels (fig. 5)
- Hack is the higher level software running on the host computer.
- The lower level Ac&Bc (APE software Channel & Backup software Channel) is hardware dependent. It runs distributed over the APE ↔

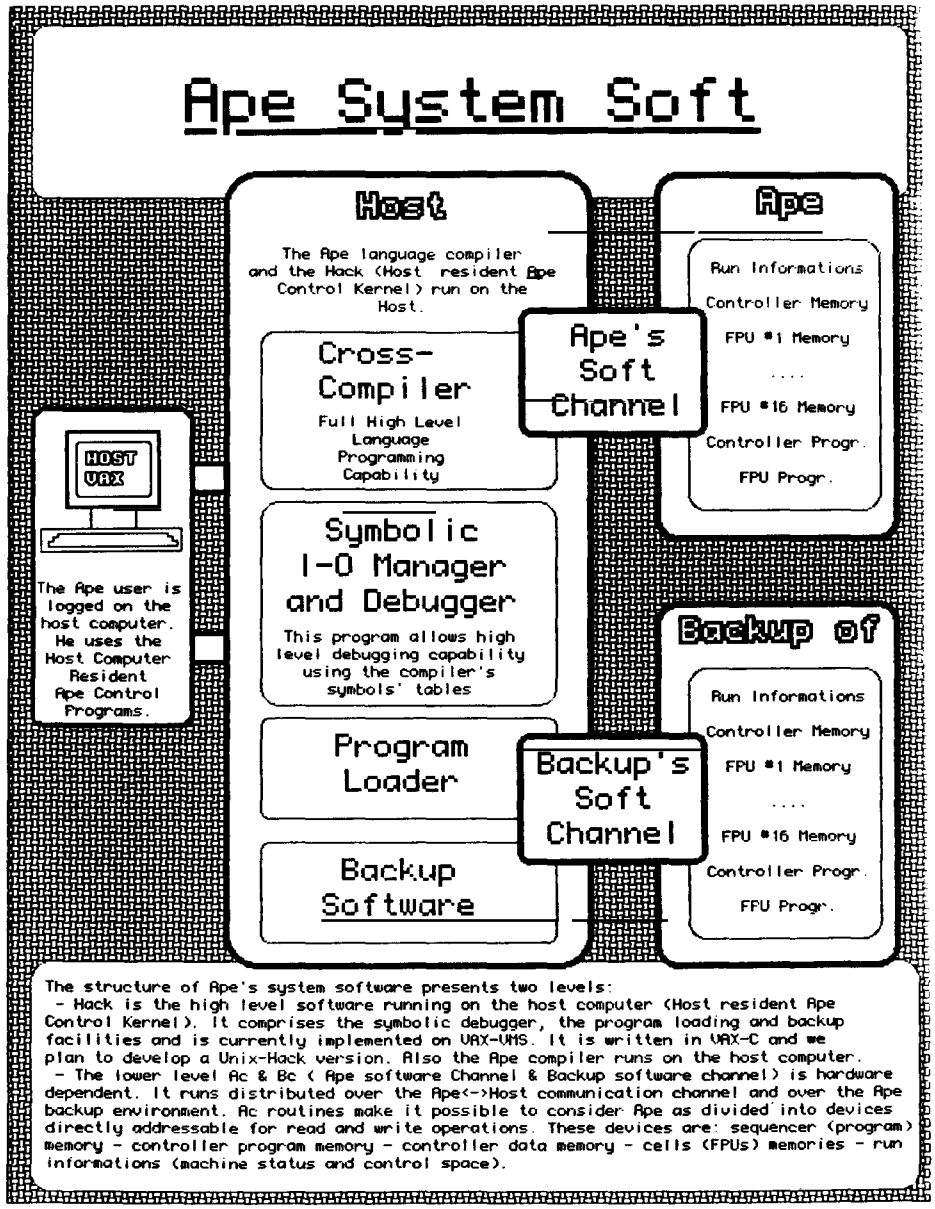

Fig. 5. The system software of APE.

host communication channel and over the APE ↔ backup environment.

Hack comprises the symbolic debugger, the program loading and backup facilities and is currently implemented on Vax-Vms. It is totally Vax C written and we plan to develop a Unix-Hack version.

The user interactive interface is provided by the Hack command interpreter supporting the Hack command language. The user can also easily incorporate calls to Hack routines in host programs. For instance a Vax-Fortran or Vax-Dcl program can call Hack's routines to run a program on APE and wait for APE run completion, and symboli-

cally access APE data for further use.

The symbolic debugger can access variables and arrays either by name or by address. It can also start programs at given labels. Comparison of symbolically selected data on APE memories and corresponding data in backup save-sets is also possible.

Monitoring and machine status display routines are also available. The Hack command language is easily extendable, as new features are needed.

APE is network accessible, since Hack is integrated in the host computer environment.

Hack is independent of the channels' hardware as it works calling the lower level Ac&Bc.

Ac routines make it possible to consider APE as divided into devices directly addressable for read and write operations. These devices are:
- sequencer (program) memory,
- controller program memory,
- controller data memory,
- cells' memories,
- machine status and control space.

Ac allows monitoring, starting and stopping of APE by reading and writing status registers and control space locations of the machine.

APE backups are structured by a similar architecture and are managed by Bc. This similarity allows Hack to access data resident on APE or on backup save sets in the same fashion.

Hack commands reconfigure Ac&Bc calls to match the actual APE hardware configuration when memory size and number of cells vary.

5. Physics applications and conclusions

The four cell prototypes have been used since september 1986 to perform lattice simulations of pure gauge systems. In particular, the glueball spectrum has been analysed, yielding results for the masses of the 0^{++} and 2^{++} states, published elsewhere [3]. The typical source code used in the simulation is about 1500 lines long, while the size of the executable codes is 180 Kbytes (or 11 Kwords). The time required to update one link is 40 μsec. This corresponds to an average speed of 70% of the theoretical peak speed. Simulations have been performed on lattices of size $10^3 * 32$, $12^3 * 32$ and $16^3 * 32$, the last lattice size almost saturating the present memory size of 32 Mbytes. More than 20000 lattice iterations have been performed in the three cases. This has required a total CPU time of about 450 hours. For comparison, a similar calculation would require about 500 CPU hours on a CRAY 1XMP supercomputer.

Overall system reliability has been very high. No system crashes have ever occurred during production runs that have lasted about 20 days.

In summary, a special purpose processor has been built and is being used for theoretical physics simulation, while application in other fields (e.g. signal processing) are being considered.

Performance of the system is comparable to that obtainable on state of the art supercomputers, for the specific application for which APE has been conceived, while development costs are roughly one order of magnitude lower than the tag price of a supercomputer.

Key features of the machine are the development of an optimized (but not over-specialized) architecture, the use of recently made available floating point chips and, possibly most important, the development of a high-level programming environment reflecting the structure of the machine in an user friendly fashion.

References

[1] P.M. Farran et al., Proc. of the Conf. on Computing in High Energy Physics, eds. L.O. Hertzberg and W. Hoogland (North-Holland, Amsterdam, 1986).
[2] The APE collaboration, Phys. Lett. to be published.