PERCOLA : A SPECIAL PURPOSE PROGRAMMABLE 64-BIT FLOATING-POINT PROCESSOR

Jean-Marie NORMAND

Service de Physique Théorique, CEN-Saclay
91 191 Gif-sur-Yvette Cedex, France

1. INTRODUCTION

The computer PERCOLA is designed for lengthy numerical simulations on a percolation problem in Statistical Mechanics of disordered media. The project that the computer is engaged on at present is intended to improve the true values of critical indices characteristic of the behaviour of electrical conductivity at percolation threshold in a system of random resistors. The architecture is based on an efficient highly iterative algorithm to compute the electrical conductivity of random resistor networks. This computer runs programs of percolation problems considered 10 percent faster than the Cray X-MP with the same 64-bit floating-point precision. Operating since May 1987, months of calculation have already been performed.

Although best suited to a class of algorithms, the processor includes a powerful 32-bit integer random number generator and has many characteristics of general purpose 64-bit floating-point microprogrammable computers that can run programs for various type of problems with a peak performance of more than 25 Mflops. This high computing speed is not the result of one all-powerful feature, but a balance among several factors including supercomputer derived features such as mainly : concurrent functional units separately controlled from independent microcode fields, distinct buses for data, addresses and instructions, flexible and powerful address generators for matrix processing and an advanced pipeline design implementing high performance VLSI Weitek's and Analog Devices components.

2. PHYSICS PROBLEM AND EQUATIONS

The percolation concept allows systems composed of large numbers of possibly interconnected objects to be described statistically. The number of objects and interconnections govern whether communication over long distances is feasible or not. Between the two categories, there is a precise transition threshold, termed the percolation threshold. The design of PERCOLA is optimized to study the critical behaviour at percolation threshold of the electrical conductivity of a random resistor network. The electrical resistances r of each bound are random variables taking two values, r_1 with probability p and r_2 with probability 1-p. Two kinds of problems are considered : a mixture of conductor-insulator ($r_1=1$ and $r_2=\infty$) where above the treshold p_c the conductivity behaves as $(p-p_c)^t$ [1]; and a mixture of superconductor-conductor ($r_1=0$ and $r_2=1$) where as p goes to p_c from underneath the conductivity diverges as $(p_c-p)^{-s}$ [2]. The aim is to increase by an order of magnitude the accuracy with which the critical indices t and s are presently known in two and three dimensions and also to explore the four-dimensional case.

The algorithm considered is based on the calculation of the electrical conductivity using the so-called strip method and the transfer matrix ideas [1]. To simulate an infinite medium in a d-dimensional space, only one dimension (the longitudinal one of size L) is made as large as possible, while finite size scaling arguments are used to take into account finite size effects in other directions (the transverse ones of size l). For a given type of lattice (e.g. hypercubic) and a transverse cross section with $N=l^{d-1}$ sites, the resistor network (a lxL strip in d=2, a lxlxL bar in d=3,...) is built by adding bounds one by one, see Fig. 1. At each step of this procedure, the system is described as an N-access black box characterized by an NxN impedance or admittance matrix **Z**. The resistance ρ of each new link is chosen at random. Then an

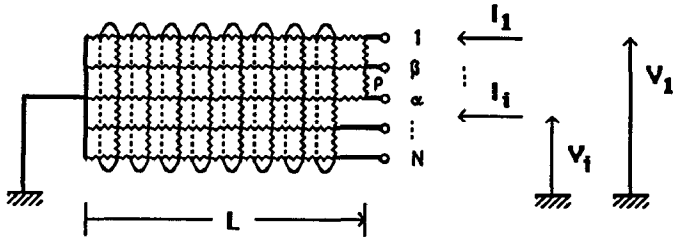

Fig. 1. A 2-dimensional strip of a random resistor network, with periodic boundary conditions from top to bottom. Potentials V and currents I are related through V=ZI.

exact calculation, using Kirchoff's laws, yields the matrix Z' of the new system in terms of the previous matrix Z and the resistance ρ of the bound added. This non-linear matrix relation $Z'=f(Z, \rho)$ has to be iterated to build a very long strip in order to reach the desired accuracy. Indeed, the convergence towards the limit for an infinite strip is like $1/\sqrt{L}$. An increase by a factor ten of the present result precision requires lengths L of order 10^9, a hundred times longer than those considered up to now. The calculation is performed for different transverse sizes l at percolation threshold. To obtain significant results, the transverse sizes l, and therefore also $N=l^{d-1}$, must be large enough. Values of N up to 256 are sufficient to study the 2 and 3-dimensional cases and even a first approach for d=4.

This algorithm leads to a highly iterative computation organized in several nesting loops : a loop over the L slices of one step in the longitudinal direction of the strip ; a loop over the d.N bounds of a slice ; for each bound : a choice of its resistance ρ from a random number generator algorithm and a loop over the update of N(N+1)/2 matrix elements of the symmetrical matrix Z according to $Z'=f(Z, \rho)$. In the most complicated case (the addition of a transverse bound between sites α and β in a superconductor-conductor mixture) this updating equation reads :

$$Z'_{ij} = Z_{ij} - Z_{NN} - \frac{(Z_{i\alpha} - Z_{i\beta})(Z_{\alpha j} - Z_{\beta j})}{(Z_{\alpha\alpha} - Z_{\alpha\beta} + Z_{\beta\beta} - Z_{\beta\alpha} + \rho)} \qquad (1)$$

provided that $Z_{\alpha\alpha} - Z_{\alpha\beta} + Z_{\beta\beta} - Z_{\beta\alpha}$ be non-zero, otherwise Z'=Z.

On the whole, such a calculation represents L.d.N random numbers and $L.d.N^2.(N+1)/2$ computations of one matrix element. These figures take huge values for the simulations considered, e.g. respectively 8. 10^{10} and 6.56 10^{13} for d=2, N=40 and $L=10^9$. Furthermore, these calculations have to be performed with a high precision in order to be able to distinguish the resistance ρ of order one from the matrix elements of order L.

3. GENERAL REQUIREMENTS FOR THE DESIGN

The idea of special purpose computers is to optimize the architecture to the problem considered. The design of a CPU then arises from compromises between several choices and requirements often strongly dependent of each other. Let us emphasize the main ones. 1) The choice of a computation scheme, i.e. a splitting of the equations to be solved into a sequence of elementary operations, elementary with respect to the functional units available. 2) The choice of computing speed taking into account the amount of time one can or wishes to spend for solving the problem. 3) The choice of technology, and especially a choice of the components which perform the main arithmetic or logic operations. 4) The choice of the type of architecture, i.e. monoprocessor, parallel, pipeline, ... 5) A constant care for checking, this must be a "leitmotiv" from the design to the operation. 6) A will to enlarge the operation capabilities of the processor. Although this seems to be inconsistent with the idea of specialization, this approach is justified for several reasons. A first one is the regard for checking mentioned previously, which leads to introduce computation capabilities in view of this only end. A second reason is to be able to cope with unexpected numerical problems which may arise from larger simulations than the ones already performed. Finally, a third reason is related to the valuation which has to be done between the interest of the particular problem to be solved and the amount of material and above all human investment required for the design and the realization of a high level computer.

To reach our aim for the percolation problem, five conditions are essential :

- *Computation speed.* Thousand hours of computation time on a Cray 1S should be necessary to get the accuracy we want. In order that the total amount of computing time remains reasonable, one has to realize a computer giving a performance equal to that of a Cray 1S, i.e. about 100 ns per update of one matrix element.
- *Precision.* Only 64-bit floating-point arithmetic fits with the high precision requirements (much larger than $L=10^9 \approx 2^{30}$) and with the wide dynamic-range needs.
- *Data storage.* At least, the N^2 (with $N \leq 256$) matrix

elements have to be stored in a high performance RAM in order to be able to read and to write one matrix element per updating cycle.

- *Random number generation.* This is a crucial point for all lengthy simulations which require a huge number of random values.

- *Nesting loop control.* At least, three nesting loops are needed.

Let us emphasize that these requirements (apart from the data memory size which is rather small) are quite general for a large number of scientific calculations. This enhances the interest of PERCOLA project since it appears as a good laboratory to acquire a know-how. The main requirement is to perform at a very high speed 64-bit floating-point arithmetic. Actually, it is the recent commercial availability of very fast 64-bit floating-point VLSI components that made the realization of PERCOLA feasible. We implement the first available chips of this family, namely the Weitek's 64-bit ALU (WTL 1065) and multiplier (WTL 1064). These two chips produce in pipeline mode one new 64-bit result every 120 ns and 480 ns, respectively. This high maximum processing speed allows us to reach our aim of computation time with a monoprocessor architecture updating one new matrix element every 120 ns. This processor has a high degree of parallelism of low level, i.e. its design is optimized by implementing several independent functional units (ALU's, multiplier, address generators, sequencer, random number generator,...) working in parallel. The data processing unit has a synchronous pipeline architecture. Indeed, the maximum throughput of Weitek's components is obtained in this operation mode. Furthermore, the most iterative part of the algorithm considered can easily be vectorized as shown below. A TTL technology has been implemented since Weitek's components have TTL compatible inputs and outputs. Furthermore, this technology offers several advantages with respect to other ones like ECL which could be considered for high speed logic : a greater number of components available, lower power dissipation, simpler to interface with a host computer of the same technology and finally easier to implement although special cares must be taken for running at high speed. We use advance Schottky (AS) and Fairchild Advanced Schottky (FAST) integrated circuits. Most of control logics are performed by very high speed programmable array logic devices (MMI's PAL series B).

The architecture is optimized for the most time consuming part of the calculation, i.e. the updating of **Z** through Eq. (1). Having in mind a pipeline architecture, and taking into account the symmetrical character of the matrix **Z**, the updating procedure of this matrix is decomposed into five steps. Two scalar steps :

1) For given α and β, the computation of the denominator :

$$D = Z_{\alpha\alpha} - Z_{\alpha\beta} + Z_{\beta\beta} - Z_{\beta\alpha} + \rho, \qquad (2)$$

where ρ is determined from a random number. Meanwhile, one checks if $Z_{\alpha\alpha}-Z_{\alpha\beta}+Z_{\beta\beta}- Z_{\beta\alpha}$ is positive. If it is true, the calculation proceeds, otherwise a new bound α' β' is considered.

2) Computation of the inverse square root of D : $S=1/\sqrt{D}$.

Two pipeline steps :

3) A loop, called pipeline 1, over the computation of the N components of a vector **V** :

$$V_i = (Z_{i\alpha} - Z_{i\beta}) \times S, \qquad i = 1, ..., N. \qquad (3)$$

4) A loop, called pipeline 2, over the updating of N(N+1)/2 matrix elements of **Z** :

$$Z'_{ij} = (Z_{ij} - Z_{NN}) - (V_i \times V_j), \quad i = 1, ..., N \; ; \; j = 1, ..., i. \qquad (4)$$

A last scalar step :

5) The accumulation of the diagonal matrix element Z_{NN} :

$$R' = R + Z'_{NN} \cdot \qquad (5)$$

In the first place, one has to optimize the pipeline steps of the calculation. Starting from the equations above, we successively consider the needs and the solutions adopted for data processing, address generation, sequencing tasks, random number generation and communication with the external word. It is emphasized that although optimized for a class of algorithms, the fully micro-programmable character of each functional unit provides a potential for future use on others problems.

3. DESCRIPTION OF THE ARCHITECTURE

64-bit floating-point data processing unit. Step 4, the longest one, requires two subtractions and one multiplication. Following the strategy of analogic computers, to update one matrix element every 120 ns-cycle in pipeline mode, one allocates independent arithmetic units running at the same speed for each of the operations. Hence, there are two ALU's WTL 1065 GC and one multiplier consisting of four WTL 1064 GC chips in parallel, working sequentially. Actually, during trouble-shooting these last chips were not able to properly operate with a 60 ns-clock and they have been replaced by four multipliers WTL 1264 GCD. In pipeline mode, each ALU and the multiplier can load two operands per cycle, i.e. one 64-bit operand per 60 ns-phase, and turn out one 64-bit result per cycle, unloaded as two 32-bit words, one per phase. In order to supply operands at such a high rate,

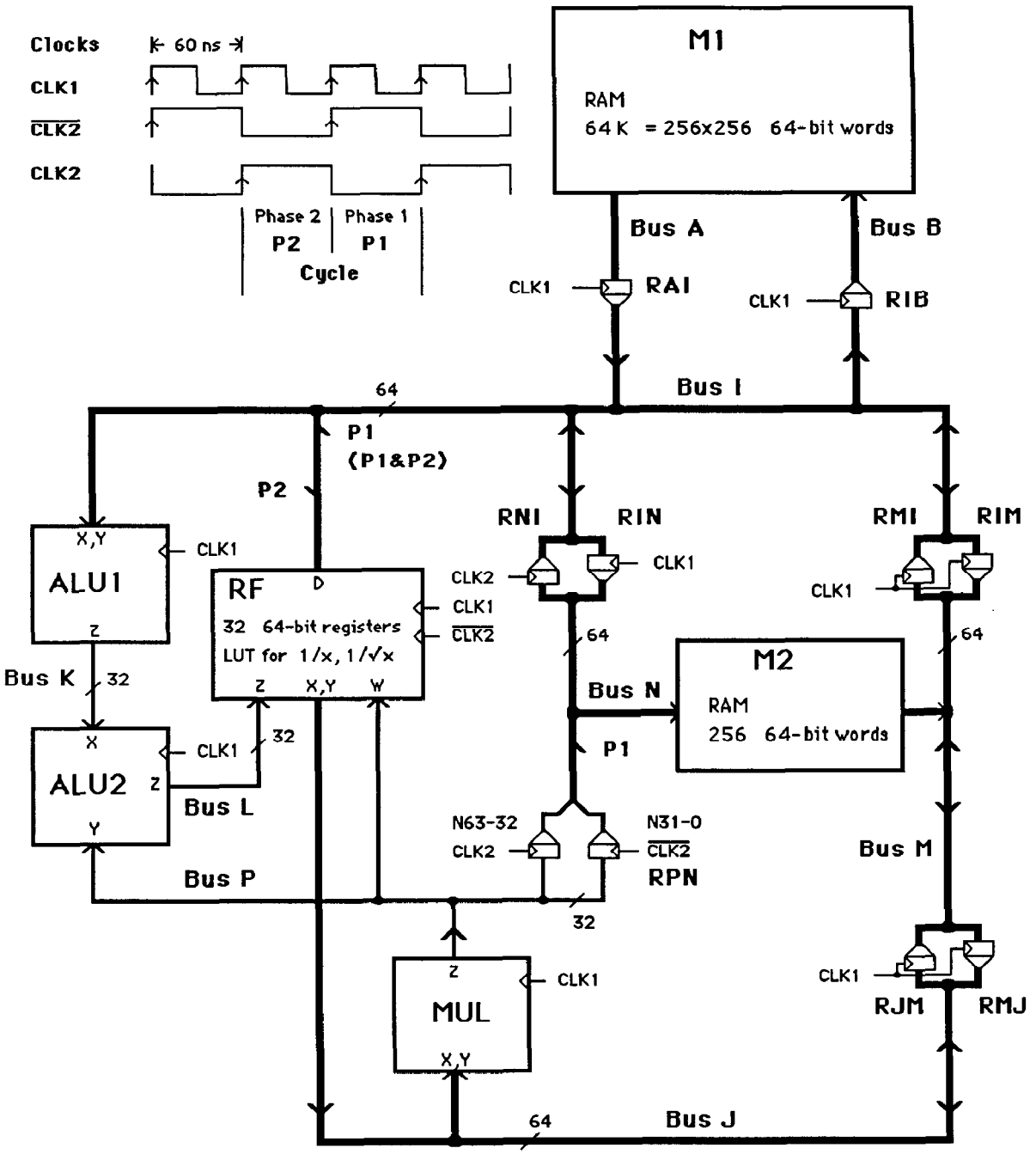

Fig. 2. Schematic diagram of the 64-bit floating-point processing unit implementing the 8 Weitek's components.

two independent fast static RAM's are introduced, both able to read or to write one 64-bit word per phase. The main 256x256=64Kx64-bit word memory M1 (64xHitachi HM6787DG-30) is devoted to the storage of the N^2 matrix elements of **Z**. The 256x64-bit word memory M2 (16xCypress CY7C122-25) provides storage for the N components of the vector **V**. Both memories M1 and M2 can read or (exclusive) write one 64-bit word per phase. The initialization of inverse square root required for step 2 is obtained from two 32-bit Weitek's register files WTL 1066 GC, working in parallel to handle 64-bit data. These two chips denoted as RF, also form a multiple port 32x64-bit word RAM which provides the necessary data bandwidth to fully utilize Weitek's pipelined floating-point arithmetic chips and to interface to data buses. Indeed, RF has a read port XY outputing one 64-bit word per phase while 64-bit words can be written through 32-bit write ports W and Z in two phases, fitting the output data flow of the ALU's and multiplier. Through the bidirectional 64-bit port D one can read and write one 64-bit word per cycle. This processing unit is designed to normally handle 64-bit floating-point operands in conformance with the requirement of FAST IEEE Standard 754, Version 10.0.

Nevertheless the whole set of Weitek's functions and mode controls is available from the microcode.

The architecture presented in Fig. 2 synthesizes all requirements previously listed : it optimizes the calculating time for pipelines 1 and 2 ; it enables a fast execution of scalar parts of the calculation, especially the computation of $1/\sqrt{x}$ (or $1/x$ and $\sqrt{x}$ if needed) by an iterative Newton-Raphson procedure involving RF, MUL and ALU2 ; it offers a large variety of data-paths for cross-checks (in "interface mode" as well as in "computing mode") and for other future applications. Let us emphasize that the control mode implemented for data-paths allows to take the best advantage of the concurrent processing and storage units and furthermore enlarge the computing capabilities. For each of buses I, J, M and N, a microcode field specifies every cycle which one of the possible data sources (all of them are available) is enable during independently each phase. Then, the several receiving devices on a same bus can concurrently be loaded according to their own microcode instructions. The pipeline stage registers systematically store data at the rising edge of their clock. A logical unit ensures a hardware conflict management on these buses.

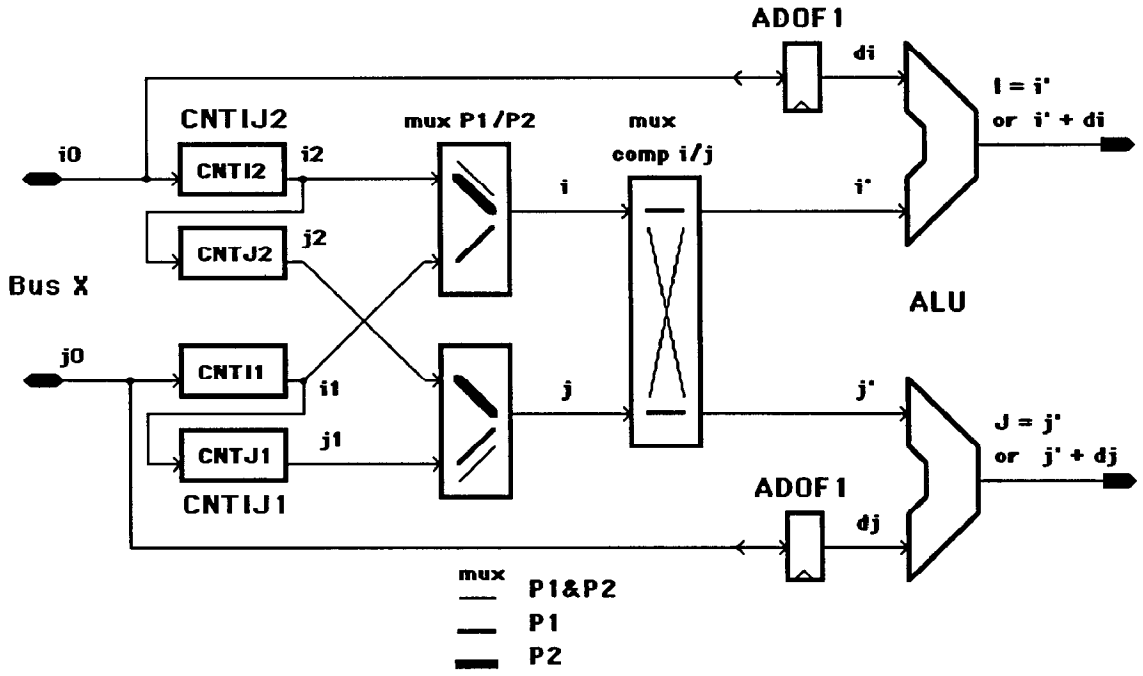

Fig. 3. Schematic diagram of the matrix address generator for memory M1.

Address generation. The algorithm considered requires high performance address generators for memories M1 and M2 optimized for matrix and vector addressing. The idea is to address any ij-matrix element by the concatenation of the two words i and j. Then for M1, a distinct processing of each index i and j (respectively row and column) allows several fast addressing modes : -Vectorial mode, i.e. a scanning of a row or a column. This mode is required for pipeline 1, where α and β are fixed in Eq. (3). -Triangular mode, i.e. a scanning of the lower or upper triangle, first diagonal included, of a square matrix. This mode allows in pipeline 2 to take into account the symmetrical character of the matrix Z, thereby reducing the computing time by a factor of almost two. Then, it is convenient to only store the triangle considered in the matrix. This implies to fetch any matrix element either at the ij or the ji address according to wether it belongs or not to the triangle stored. This mechanism is necessary in pipeline 1 for the addresses $i\alpha$ and $i\beta$, where α and β are fixed while i varies. -Rectangular mode, i.e. a scanning of a rectangular matrix is also available for general purpose. -Random mode, i.e. a scanning over a particular sequence of addresses. This mode is required for the loop over bounds of a strip slice, especially the sequence of transverse bounds $\alpha\beta$. This problem has been settled thanks to an independent RAM, called M3, which can be initialized from the host computer. Then, M3 is addressed by an internal counter of the sequencer, thereby handling loops over any loaded sequence of addresses. In addition to the address $\alpha\beta$, the calculation step 1 requires the addresses $\alpha\alpha$ and $\beta\beta$, which can conveniently be obtained from $\alpha\beta$ through a broadcasting mechanism over the two words α and β. For M2, a vectorial addressing mode is required in pipeline 1. For pipeline 2, one needs a triangular addressing mode, i.e.

alternately addressing the i and j components of V in concordance with the triangular scanning of M1. A similar rectangular mode is also available.

To reach the maximum speed performances of data processing units requires that the address generators of each memory M1 and M2 outputs one new address every 60 ns-phase. Furthermore, during pipeline 2 the address generator of M1 has to alternately produce a reading and a writing address of a matrix element, the lag between these two address sequences being the updating time for one matrix element. This leads to an architecture with two independent address generators, one for each of the two phases of a 120 ns-cycle. A similar structure has also been adopted for the address generator of M2.

Since no commercially available integrated device meet our requirements, the address generators have been specially designed for the computer, implementing some fifty chips. The schematic diagram of the address generator for M1 is given in Fig. 3. It is composed of four cascaded stages, each of them separately controlled every cycle by its own microcode field . The first stage is made of two identical and independent pairs of counters. Each pair of counters generates one address per cycle and each counter controls one index. The second stage is a phase multiplexor which outputs one address per phase. This

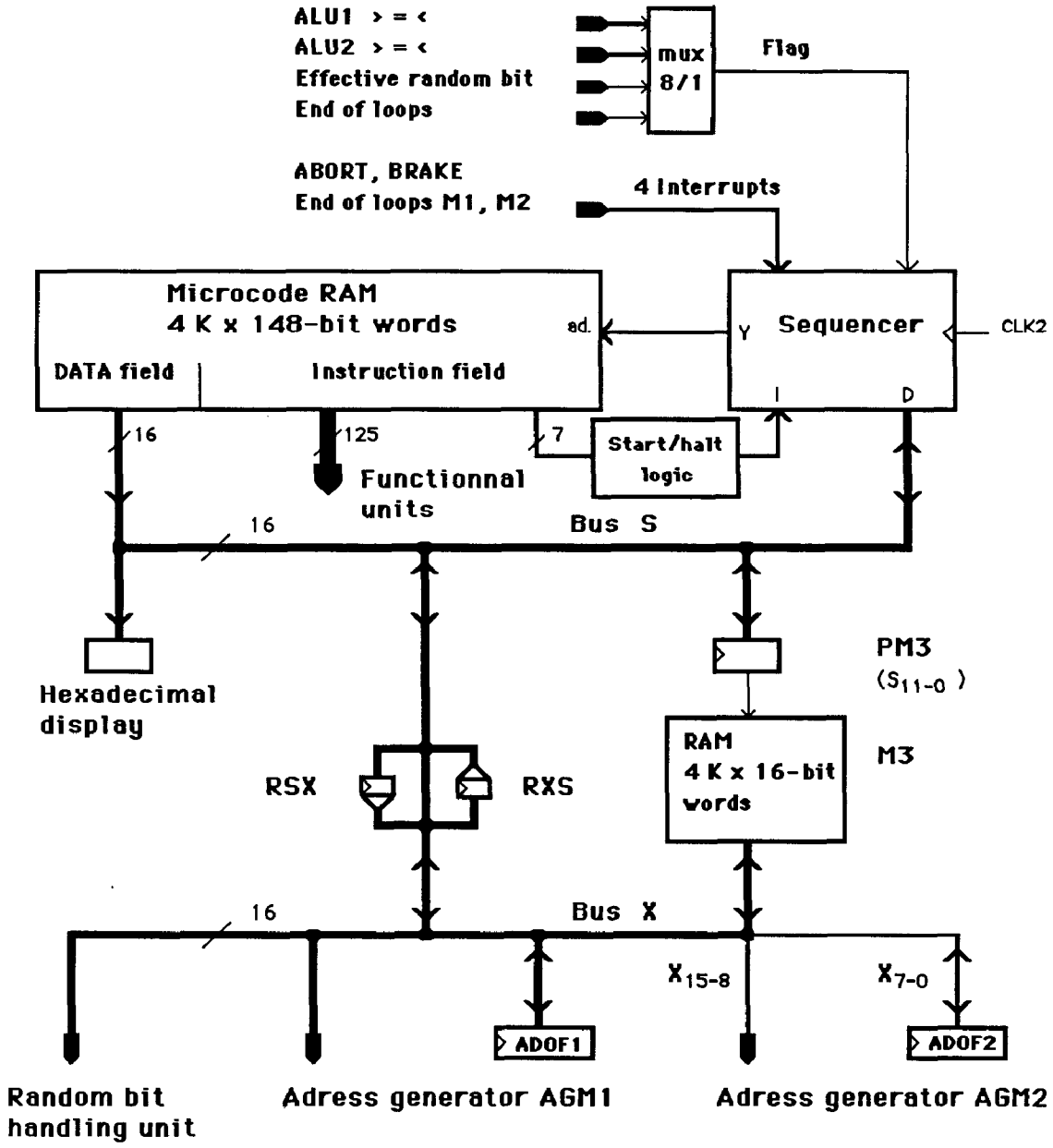

Fig. 4. Schematic diagram of the sequencing unit based on the Analog Devices sequencer ADSP 1401.

stage also provides a broadcasting mechanism which, from a stored ij address, generates both addresses ii and jj. The third stage is an index multiplexor associated with a comparator of input indices. This stage allows index permutations, either systematic or according to the comparison between both input indices. The last step provides offset capabilities separately for each index. Since the loops executed by the counter stage always ending to the address 0, the address offset allows to fully utilize the memory space available. The design and the operating mode of the address generator fo M2 are similar, although simpler. These address generators are initiated through the 16-bit bus X, see Fig. 4. This bus can be driven from a 16-bit data microcode field, the data port of the sequencer or the 4Kx16-bit word static RAM M3 4xInmos IMS1423-25). These several data sources provides as much additional addressing modes. The six internal ports of the 32-register RF are each directly addressed from the microcode.

Sequencing unit. The schematic diagram of this unit is shown in Fig. 4. Each 120 ns-cycle, a 148-bit instruction specifies the operations to be performed by all elementary functional units of the processor. The set of these instructions is stored in the microcode memory which is an independent 4Kx148-bit word static RAM (37xInmos IMS1423-25) having its own data and address buses. The micoprogram sequencer provides the appropriate microprogram addressing to support programming requirements, such as mainly several nesting loops and conditional branchings (according to the random bit and the result of a comparison in step 1 of the calculation). The powerful VLSI program sequencer Analog Devices ADSP-1401 KD meets our requirements of speed and functionalities. The whole set of instructions provided by Analog Devices for this chip is available. A 16-bit bidirectional data port, connected to the bus S, permits loading or dumping of all internal registers and supplies direct or indirect jump addresses. Four external maskable interrupts are implemented : an ABORT (from host computer) and a BRAKE (from microcode), both inducing the execution of a programmed dump of the sequencing unit into memory M3 ; an end of loop signal for each of address generators for M1 and M2. An external flag input can control conditional instructions. This pin is connected to the output of a 1 of 8 bit selector controlled from microcode. The eight inputs of this multiplexor are the following : two sets of three bits coming from the status registers of each Weitek's ALU's (each bit corresponds to one of the comparisons <, = and >) ; the effective random bit outputs by the random bit handling unit (see latter) ; and an end of loop signal which is the OR of both end of loop signals for M1 and M2.

A microcode field provides many instructions of the type source-destination to transfer data between the several units shown in Fig. 4. Most of these instructions transfer one 16-bit word per cycle. Some instructions executes two simultaneous transfers per cycle, e.g. RXS into M3 and SEQ into PM3. Furthermore, when a one-transfer instruction only involves bus S, bus X is driven by M3 as default option.

Random number generation. We need a random bit b such that b=1 with probability p and 0 with probability 1-p, where p is a parameter within the range [0,1]. To treat anisotropic media requires a possible change of p at any random number request. Furthermore, in the most simple case (the addition of a longitudinal bound at site α in a superconductor-conductor mixture) only one matrix element of **Z** has to be updated according to $Z'_{\alpha\alpha}=Z_{\alpha\alpha}+\rho$. Then, to perform a loop over the updating for these longitudinal bounds at the maximum pipeline speed demands one new random bit per 120 ns-cycle. Last but not least, it has already been noted that we need a huge number of random values, more than 10^{10}. Actually, the ability to generate satisfactory sequences of random numbers is one of the key links between Computer Science and Statistics. For reasons of speed, simplicity and repeatability, random numbers are produced by a given algorithm, chosen in such a way that the results pass extensive tests of randomness. Indeed, if the maximum period can often be analytically determined, it is not the same with correlations. Hence, for any "best" procedure, there is no certainty that either larger simulations or other types of simulations will not find the algorithm unsatisfactory.

To produce the random bit b, one generates a "uniform" sequence of random numbers $\{x_i\}$. Each of these numbers is compared with the threshold P equal to the renormalized value of p with respect to the range of variation of x_i's. The result of this comparison is the random bit b, either 0 if $x_i \geq P$ or 1 if $x_i < P$. The random numbers generator is of lagged-Fibonacci type handling integers mod 2^n . From an initial set of values $x_1, x_2, \ldots, x_r$ and two lags r and s with r>s, one generates successive elements by the recurrence :

$$i > r \qquad x_i = x_{i-r} + x_{i-s} \mod 2^n. \tag{6}$$

This algorithm has been chosen for its facility of implementation (only one addition is needed), and above all for its statistical properties [3]. The precision we want for p requires to work with 32-bit integers. We consider three couples of (r,s), i.e. (17,5), (31,13) and (55,24),

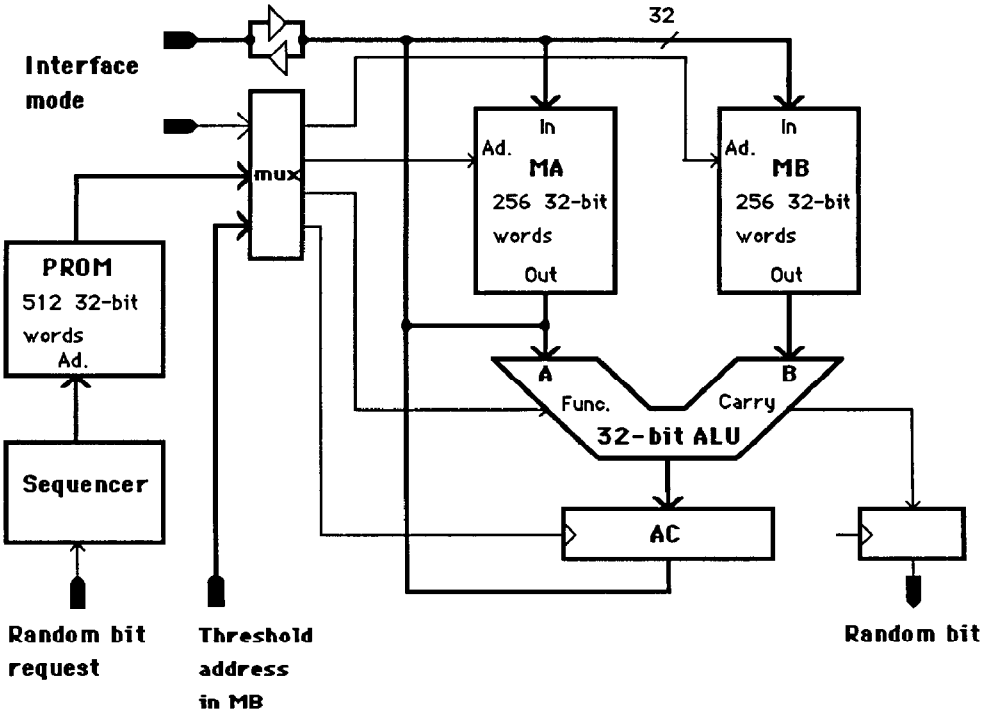

Fig. 5. Schematic architecture of the random bit generator : an independent 32-bit integer processor.

corresponding to three generators of maximum period $(2^r-1)2^{n-1}$, respectively 2.8 10^{14}, 4.6 10^{18} and 7.7 10^{25}. Comparison between simulations performed with these three generators provides useful informations about the statistical properties of the sequences generated by this algorithm, especially in the new field of more than 10^{10} random values.

The schematic diagram of the random bit generator is given in Fig. 5. The computation cycle of one random bit is split into three 40 ns-phases. In phase 1 two identical 256x32-bit word static RAM's (16xCypress CY7C122-15) MA and MB simultaneously provide the operands x_{i-r} and x_{i-s} to be added in a 32-bit integer ALU (full look-ahead adder with 8x74AS181A and 2x74AS882). The result x_i is stored in a 32-bit register AC. In phase 2 the ALU executes a comparison between x_i and the threshold P previously loaded in MB. The result of this comparison is the random bit b. During the last step 3 the value x_i is simultaneously written in place of the no longer needed value x_{i-r}. Indeed, at each cycle one only requires the last r values generated of the x's to proceed the recursion. The sequencing tasks of this independent 32-bit arithmetic unit are defined by a microcode in registered PROM (4xMMI 63RA481A). A cyclic counter, pre-loaded according to the parameters (r,s) chosen, produces the PROM addresses.

To handle site percolation (where the independent random variables are occupation probabilities of sites instead of bound variables) requires a dedicated logical unit which generates an effective random bit for each bond in terms of the occupation probabilities of sites. These latter bits are produced by the random bit generator described above, and stored in a 256x4-bit static RAM.

Communication with the external word and start-up and halt controls. Communication with the special purpose computer is established from a host computer by two interfaces, one in each of these computers, connected through cables. The processor has two operating modes : the "computing mode" controlled by the microcode and the "frozen mode", where all processor clocks (except the sequencing unit clock) are simultaneously halted at a definite time. In "frozen mode" the master of data and address buses can be either the processor or the interface with the host computer. This last case is the "interface mode" which, answering our constant care for checking, provides the access to any memory location, a memory cell as well as a simple register (including all 64-bit pipeline stage registers, except for RMJ, the address pointer of each memory and a status register). The processor can be (re)started either at any absolute address of the microcode memory, or sequentially at the address stored in the program counter of the sequencer. The halt of the processor, i.e. its setting into the "frozen mode", can be

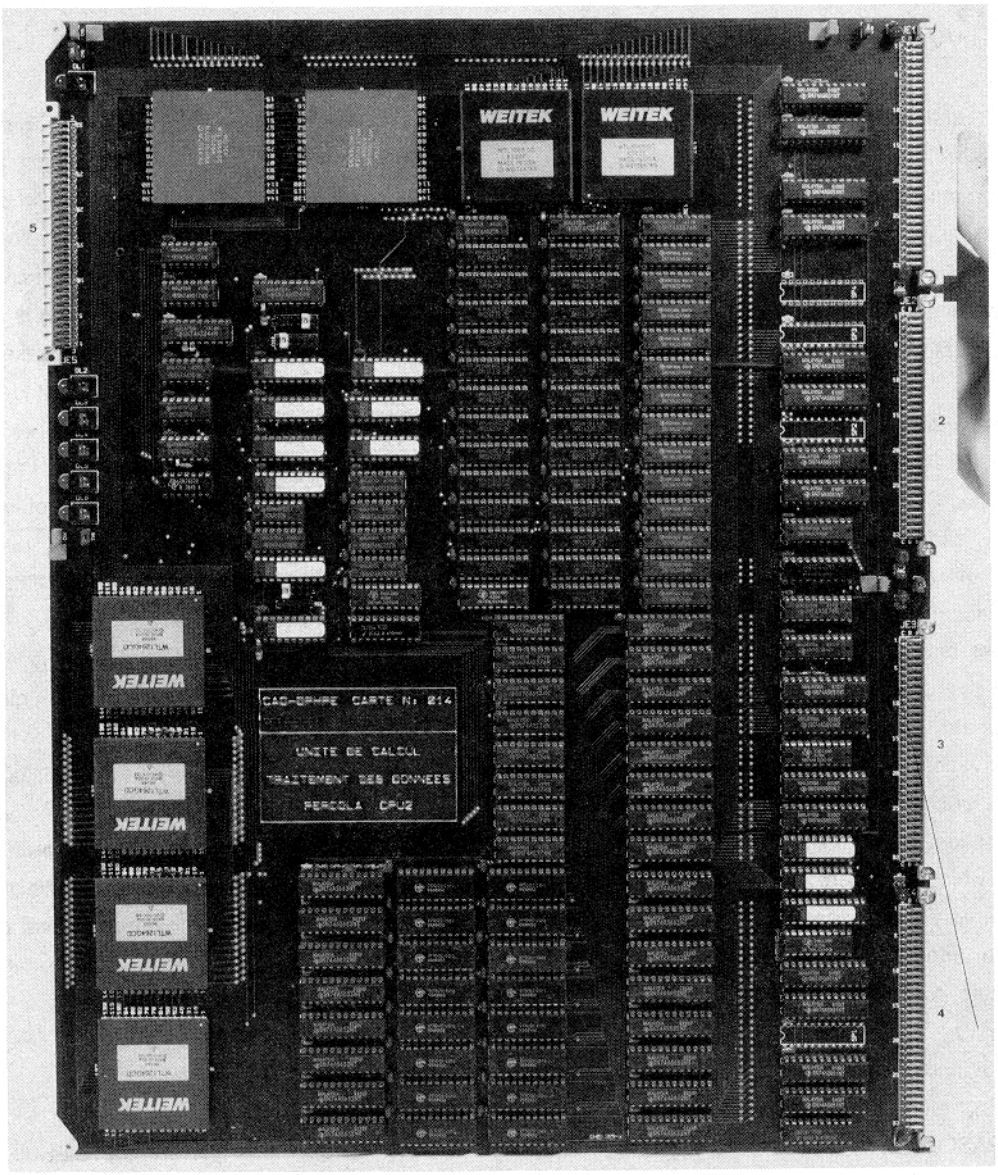

Fig. 6. Photography of the 64-bit data processing unit board (except M1), a ten layer printed circuit board of size 41 cm x 32 cm implementing the 8 Weitek's components (144-pin PGA).

triggered by a microcode bit, by the detection of a maskable Weitek's error (e.g. overflow and underflow) or at any time by the host computer. The "frozen mode" is an original feature which is important for checking and program debugging (with the sequential start-up facilities, a step by step execution of programs is possible).

The host computer is the ISADORA built at Saclay from VME standard boards based on a Motorola 68000 microprocessor with 2M byte memory, a double port VME-VMX 1M byte memory (to allow fast DMA between the host and the processor), one floppy and two hard Winchester disks to periodically save results and store microcode and interface control programs. This host is connected to the netlink "TRANSPAC" allowing its full remote control.

Assembling. Six multi-layer printed circuit boards have been realized, all of them including ground and Vcc planes. Each of the 620 integrated circuits is decoupled by a capacitor and plugged into a socket. Five boards of size 41 cm x 32 cm are inserted into the processor rack : an interface (4 layers), the random bit generator (6 layers), the memory M1 (6 layers), the sequencing unit and address generators (8 layers), and the 64-bit data processing unit (10 layers), see Fig. 6. The sixth board is the other interface (8 layers)

plugged into the VME rack of the host computer.

4. PROGRAMMING AND OPERATION

The programming of the special purpose computer is performed in assembly language. It is based on three types of information. Two of them are standard for any microprocessor, even though the processor is more elaborate, i.e. the block diagrams of each functional unit (e.g. Figs. 2-4), and the set of mnemonics for each individual control fields (e.g. the whole set of instructions provided by Analog Devices and Weitek, respectively for the sequencer and the 64-bit operators). On the other hand the third information is non-standard and due to the sophisticated pipeline architecture of the machine. Indeed, according to the different microcode fields, there exists different delays (in number of 60 ns -phases) between the cycle where the instruction is provided and the phase of its effective implementation. A programming aid is provided by programming sheets including reservation tables for all buses and the microcode fields of each functional unit. From a program written in the assembly language, a micro-assembler generates the binary microcode file which is loaded from the host computer into the microcode memory. A dissassembler is also available for debugging. In addition to percolation programs, several selective tracing routines for most of functional units are available.

To control and initialize the special purpose computer from the host, elementary loading routines have been developed in PASCAL and 68000 assembler language to access to all memory locations. Then, implementing these basic routines, two main programs, with tree-like user friendly menus have been written : one for general purpose controls of the processor, the other for an easy operation of percolation routines.

First results, concerning critical indices which characterize the behaviour of electrical conductivity at percolation threshold in a two-dimensional system of random resistors, are going to be published [4]. In addition, one gets useful information about the random number generator considered in the new field of more than 10^{10} random values. These results are obtained from more than six months of computation. Optimize programs for the percolation problem, of about one thousand micro-instructions, run ten percent faster than the same algorithm written in FORTRAN on a Cray X-MP used as a mono-processor. The 64-bit scalar functions $1/x$ and $1/\sqrt{x}$ are respectively performed in 7.32 μs and 8.40 μs.

4. CONCLUSION

The improvement in VLSI high speed components and the implementation of original architectures allow to get very large computational power for special class of algorithms with an excellent cost/performance ratio (the overall cost of PERCOLA is less than a million French francs). At the price of a non-standard programming, which is not a serious drawback for dedicated computers, one can take the best advantage of new hardware facilities. For future developments, the main PERCOLA's assets are : for data processing, as in analogic computers, a pipeline design implementing as much concurrent functional units as required by equations ; for address generation, a design well adapted to matrix element and more generally indexed data handling ; an independent random number generators.

The project PERCOLA was initiated by H.J. Herrmann [5]. Design, assembly and programming of the machine are the work (almost three years) of J.-M. Normand [6,7], who worked as project leader with M. Hajjar, who wrote his thesis [6,8]. We are grateful to the Service d'électronique et d'informatique de particules élémentaires, Centre d'Etude Nucléaires de Saclay for its hospitality and help, and more particularly B. Ollivier concerning the CAD.

REFERENCES

[1] B. DERRIDA and J. VANNIMENUS, *J. Phys.* A 15 (1982), L557 ; B. DERRIDA, D. STAUFFER, H.J. HERRMANN and J. VANNIMENUS, *J. Phys. (Paris) Lett.* **44** (1983), L701 ; B. DERRIDA, J. G. ZABOLITZKY, J. VANNIMENUS, and D. STAUFFER, *J. Stat. Phys.* **36** (1984), 31.

[2] H.J. HERRMANN, B. DERRIDA and J. VANNIMENUS, *Phys. Rev. B* **30** (1984), 4080.

[3] G. MARSAGLIA, in "Encyclopedia of Computer Science and Engineering", 2nd edition p. 1260, Van Nostrand Reinhold Co., New York (1983).

[4] J.-M. NORMAND, H.J. HERRMANN and M. HAJJAR, to be published in *J. Stat. Phys.* (1988).

[5] F. HAYOT, H.J. HERRMANN, J.-M. NORMAND, P. FARTHOUAT and M. MUR, *J. Comput. Phys.* **64** (1986), 380.

[6] J.-M. NORMAND and M. HAJJAR, *Supplément au Bulletin de la Société Française de Physique*, **65** (1987), 51 ; J.-M. NORMAND and M. HAJJAR, "PERCOLA : a programmable 64-bit floating-point processor optimized for a percolation problem", to be published (1988).

[7] J.-M. NORMAND, communication in "Algorithmes et architectures à haut degré de parallélisme" ONERA & TIM3, La Briantais (Saint-Malo), Novembre 1987, edited by INRIA ; J.-M. NORMAND, CLEFS CEA, N° 8 (1988).

[8] M. HAJJAR, Thesis, University of Paris-Sud, Centre d'Orsay (1987).