Programmable Active Memories: Reconfigurable Systems Come of Age

J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, P. Boucard

Abstract— **Programmable Active Memories (PAM) are a novel form of universal reconfigurable hardware co-processor. Based on *Field-Programmable Gate Array* (FPGA) technology, a PAM is a *virtual* machine, controlled by a standard microprocessor, which can be dynamically and indefinitely reconfigured into a large number of application-specific circuits. PAMs offer a new mixture of hardware performance and software versatility.**

We review the important architectural features of PAMs, through the example of DECPeRLe-1, an experimental device built in 1992.

PAM programming is presented, in contrast to classical gate-array and full custom circuit design. Our emphasis is on large, code-generated synchronous systems descriptions; no compromise is made with regard to the performance of the target circuits.

We exhibit a dozen applications where PAM technology proves superior, both in performance *and* cost, to every other existing technology, including supercomputers, massively parallel machines, and conventional custom hardware. The fields covered include computer arithmetic, cryptography, error correction, image analysis, stereo vision, video compression, sound synthesis, neural networks, high-energy physics, thermodynamics, biology and astronomy.

At comparable cost, the computing power virtually available in a PAM exceeds that of conventional processors by a factor 10 to 1000, depending on the specific application, in 1992. A technology shrink increases the performance gap between conventional processors and PAMs. By Noyce's law, we predict by how much the performance gap will widen with time.

Keywords— **Programmable Active Memory, PAM, reconfigurable system, field-programmable gate array, FPGA.**

I. Introduction

THERE are two ways to implement a specific high-speed digital processing task.

- The simplest is to program some general-purpose computer to perform the processing. In this *software* approach, one effectively maps the algorithm of interest onto a *fixed* machine architecture. However, the structure of that machine will have been highly optimized to process arbitrary code. In many cases, it will be poorly suited to the specific algorithm, so performance will be short of the required speed.
- The alternative is to design ad hoc circuitry for the specific algorithm. In this *hardware* approach, the machine structure—processors, storage and interconnect—is tailored to the application. The result is more efficient, with less actual circuitry than general-purpose computers require.

This work was done at Digital Equipment Corporation's Paris Research Laboratory (DEC-PRL, 92500 Rueil-Malmaison, France) from 1988 to 1994. J. Vuillemin, D. Roncin, M. Shand, H. Touati and P. Boucard were members of PRL research staff. P. Bertin was a visiting scientist from Institut National de Recherche en Informatique et en Automatique (INRIA, 78150 Rocquencourt, France).

The disadvantage of the hardware approach is that a specific architecture is usually limited to processing a small number of algorithms, often a single one. Meanwhile, the general-purpose computer can be programmed to process *every* computable function, as we have known since the days of Church and Turing.

Adding special-purpose hardware to a universal machine, say for video compression, speeds up the processor—when the system is actually compressing video. It contributes nothing when the system is required to perform some different task, say cryptography or stereo vision.

We present an alternative machine architecture that offers the best of both worlds: software versatility *and* hardware performance. The proposal is a standard high-performance microprocessor enhanced by a PAM co-processor. The PAM can be configured as a wide class of specific hardware systems, one for each interesting application. PAMs merge together hardware and software.

This paper presents results from seven years of research, at INRIA, DEC-PRL and other places. It addresses the following topics:

How to build PAMs.
How to program PAMs.
What are the applications?

Section II introduces the principles of the underlying FPGA technology.

Section III highlights the interesting features of PAM architecture.

Section IV presents some of the methods used in programming *large* PAM designs.

Section V describes a dozen applications, chosen from a wide variety of scientific fields. For each, PAM outperforms all other existing technologies. A hypothetical machine equipped with a dozen different conventional co-processors would achieve the same level of performance—at a higher price. Through reconfiguration, a PAM is able to *time-share* its internal circuitry between our twelve (or more) applications; the hypothetical machine would require different custom circuits for each, that must be physically present at all times.

We assess, in Section VI, the computing power of PAM technology, today and in the future.

II. Virtual circuits

The first commercial FPGA was introduced in 1986 by Xilinx [1]. This revolutionary component has a large internal configuration memory, and two modes of operation: in *download* mode, the configuration memory can be written, as a whole, through some external device; once in *con-*

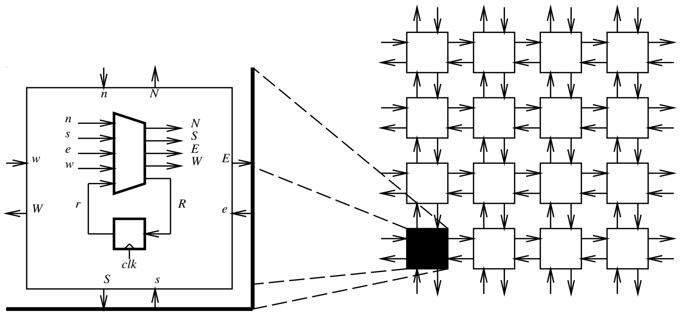
figured mode a FPGA behaves like a regular *application-specific integrated circuit* (ASIC).

To realize a FPGA, one simply connects together in a regular mesh, $n \times m$ identical *programmmable active bits* (PABs). Surprisingly enough, there are many ways to implement a PAB with the required universality. In particular, it can be built from either or both of the following primitives:

- a *configurable logic block* implements a boolean function with k inputs (typically $2 \leq k \leq 6$); its truth table is defined by 2^k (or less) configuration bits, stored in local registers;
- a *configurable routing block* implements a switchbox whose connectivity table is set by local configuration bits.

Such a FPGA implements a Von Neumann cellular automaton. What is more, the FPGA is a *universal* example of such a structure: any synchronous digital circuit can be emulated, through a suitable configuration, on a large enough FPGA, for a slow enough clock.

Some vendors, such as Xilinx [2] or AT&T [3], form their PABs from both configurable routing and logic blocks. Other early ones, such as Algotronix [4] (now with Xilinx) or Concurrent Logic [5] (now with Atmel), combine routing and computing functions into a single primitive—this is the *fine grain* approach. An idealized implementation of this fine grain concept is given in figure 1. A third possibility is to build the PAB from a configurable routing box connected to a fixed (non configurable) universal gate such as a *nor* or a *multiplexor* [6].



This PAB has 4 inputs $\langle n, s, e, w\rangle$, 4 outputs $\langle N, S, E, W\rangle$, one register (flip-flop) with input R and output r, and a combinational gate

$$g(n, s, e, w, r) = (N, S, E, W, R)$$

The truth table of g is specified by $160 = 5 \times 32$ bits.

Fig. 1. Field-programmable gate array

Each FPGA implementation can emulate each of the others, granted enough PABs. In order to make quantitative performance comparisons between the diverse significant implementations, let us, from now on, choose as our reference unit any active bit with one 4-input boolean function—configurable or not—and one internal bit of state (see section VI and Vuillemin [7]). With its five 5-input functions, the PAB from figure 1 counts for ten or so such units.

The FPGA is a *virtual circuit* which can behave like a number of different ASICs: all it takes to emulate a particular one is to feed the proper configuration bits. This means that prototypes can be made quickly, tested and corrected. The development cycle of circuits with FPGA technology is typically measured in weeks, as opposed to months for hardwired gate array techniques. But FPGAs are used not just for prototypes; they also get incorporated in many production units. In all branches of the electronics industry other than the mass market, the use of FPGAs is expanding, despite the fact that they still cost ten times as much as ASICs in volume production. In 1992, FPGAs were the fastest growing part of the semi-conductor industry, increasing output by 40 %, compared with 10 % for chips overall.

As a consequence, FPGAs are on the leading edge of silicon chips. They grow bigger and faster at the rate of their enabling technology, namely that of the static RAM used for storing the internal configuration.

In the past 40 years, the feature size of silicon technology has been shrinking by a factor $1/\alpha \approx 1.25$ each year. This phenomenon is known as Noyce's thesis; it was first observed in the early sixties. The implications of Noyce's thesis for FPGA technology are analyzed by Vuillemin [7]. The prediction is that the leading edge FPGA, which has 400 PABs operating at 25 MHz in 1992, will, by year 2001, contain 25k PABs operating at 200 MHz.

III. PAMs as Virtual Machines

The purpose of a PAM is to implement a *virtual machine* that can be dynamically configured as a large number of specific hardware devices.

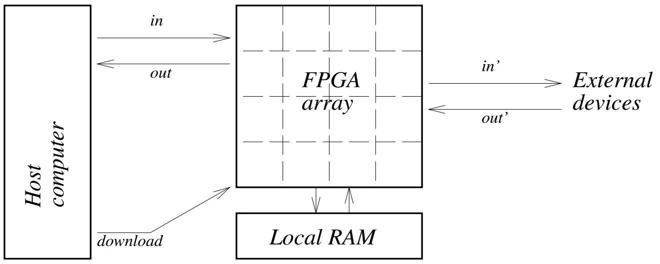


Fig. 2. Programmable Active Memory

The structure of a generic PAM is found in figure 2. It is connected—through the *in* and *out* links—to a host processor. A function of the host is to *download* configuration bitstreams into the PAM. After configuration, the PAM behaves, electrically and logically, like the ASIC defined by the specific bitstream. It may operate in stand-alone mode, hooked to some external system—through the in' and out' links. It may operate as a co-processor under host control, specialized to speed-up some crucial computation. It may operate as both, and connect the host to some external system, like an audio or video device, or some other PAM.

To justify our choice of name, observe that a PAM is attached to some high-speed bus of the host computer, like any RAM memory module. The processor can write

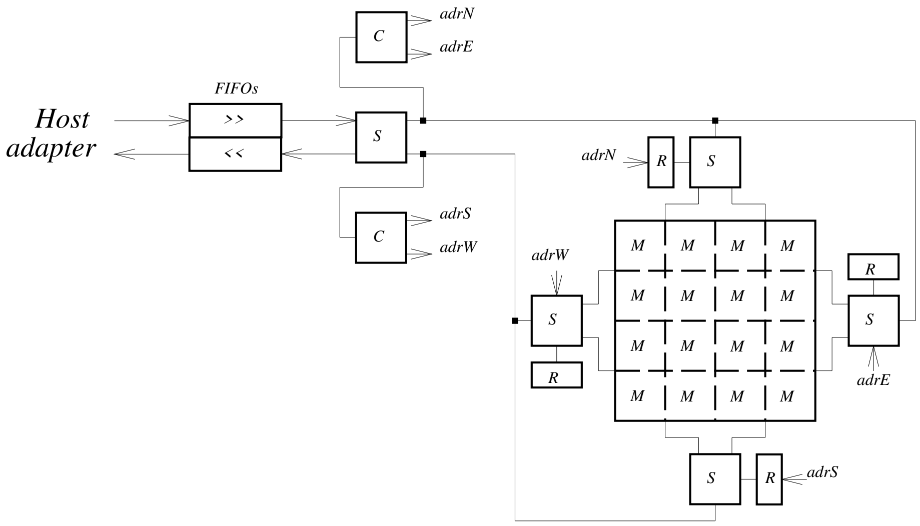


Fig. 3. DECPeRLe-1 architecture

into, and read from the PAM. Unlike RAM however, a PAM processes data between write and read instructions—which makes it an "active" memory. The specific processing is determined by the contents of its configuration bitstream, which can be updated by the host in a matter of milliseconds—thus the "programmable" qualifier.

We now describe the architecture of a specific PAM: it is named DECPeRLe-1 and will be referred to as P_1. It was built at Digital's Paris Research Laboratory in 1992. A dozen copies operate at various scientific centers in the world; some are cited as we enumerate the operational applications in section V.

The overall structure of P_1 is shown in figure 3. Each of the 23 squares denotes one Xilinx XC3090 FPGA [2]. Each of the 4 rectangles represents 1 MB of static RAM (letter R). Each line represents 32 wires, physically laid out on the printed circuit board (PCB) of P_1. A photo of the system is shown in figure 4.

The merit of this structure is to host, in a natural manner, the diverse networks of processing units presented in section V. Depending upon the application, individual units are implemented within one to many FPGAs; they may also be implemented as *look-up tables* (LUT) through the local RAM; some slow processes are implemented by software running on the host. Connections between processing units are mapped, as part of the design configuration, either on PCB wires or on internal FPGA wires.

A. FPGA matrix

The computational core of P_1 is a 4×4 matrix of XC3090—letter M in figure 3. Each FPGA has 16 *direct connections* to each of its four Manhattan neighbors. The four FPGAs in each row and each column share two common 16-bit buses. There are thus four 64-bit buses traversing the array, one per geographical direction N, S, E, W.

The purpose of this organization is to best extrapolate, at the PCB level, the internal structure of the FPGA. What we have is close to a large FPGA with 64×80 PABs—except for a connection bottleneck every quarter of the array, as there are fewer wires on the PCB than inside the FPGA. By Noyce's thesis, P_1 implements, with 1992 technology, a leading edge FPGA that should become available on a single chip by 1998.

B. Local RAM

Some applications, like RSA cryptography, are entirely implemented with FPGA logic; most others require some amount of RAM to buffer and re-order local data, or to implement specialized LUTs.

The size of this cache RAM is 4 MB for P_1, made up of four independent 32-bit-wide banks. The 18-bit addresses and read/write signals for each RAM are generated within one of two *controller* FPGAs—letter C in figure 3. Data to and from each RAM goes to the corresponding *switch* FPGA—letter S.

All the presented applications that do use the RAM operate around 25 MHz. Many utilize the full RAM bandwidth available, namely 400 MB/s. Other applications, for which RAM access is not critical, operate at higher clock speeds, such as 40MHz for RSA, and higher.

C. External links

P_1 has four 32-bit-wide external connectors.

Three of these (not represented on figure 3) link edges of the FPGA matrix to external connectors. They are used for establishing *real-time* links, at up to 33 MHz, between P_1 and external devices: audio, video, physical detectors... Their aggregated peak bandwidth exceeds 400 MB/s.

The fourth external connection links to the host interface of P_1: a 100 MB/s *TURBOchannel* adapter [8]. In order to avoid having to synchronize the host and PAM clocks, host data goes through two FIFOs, for input and output respectively. To the PAM side of the FIFOs is another switch FPGA, which shares two 32-bit buses with the other switches and controllers—see figure 3.

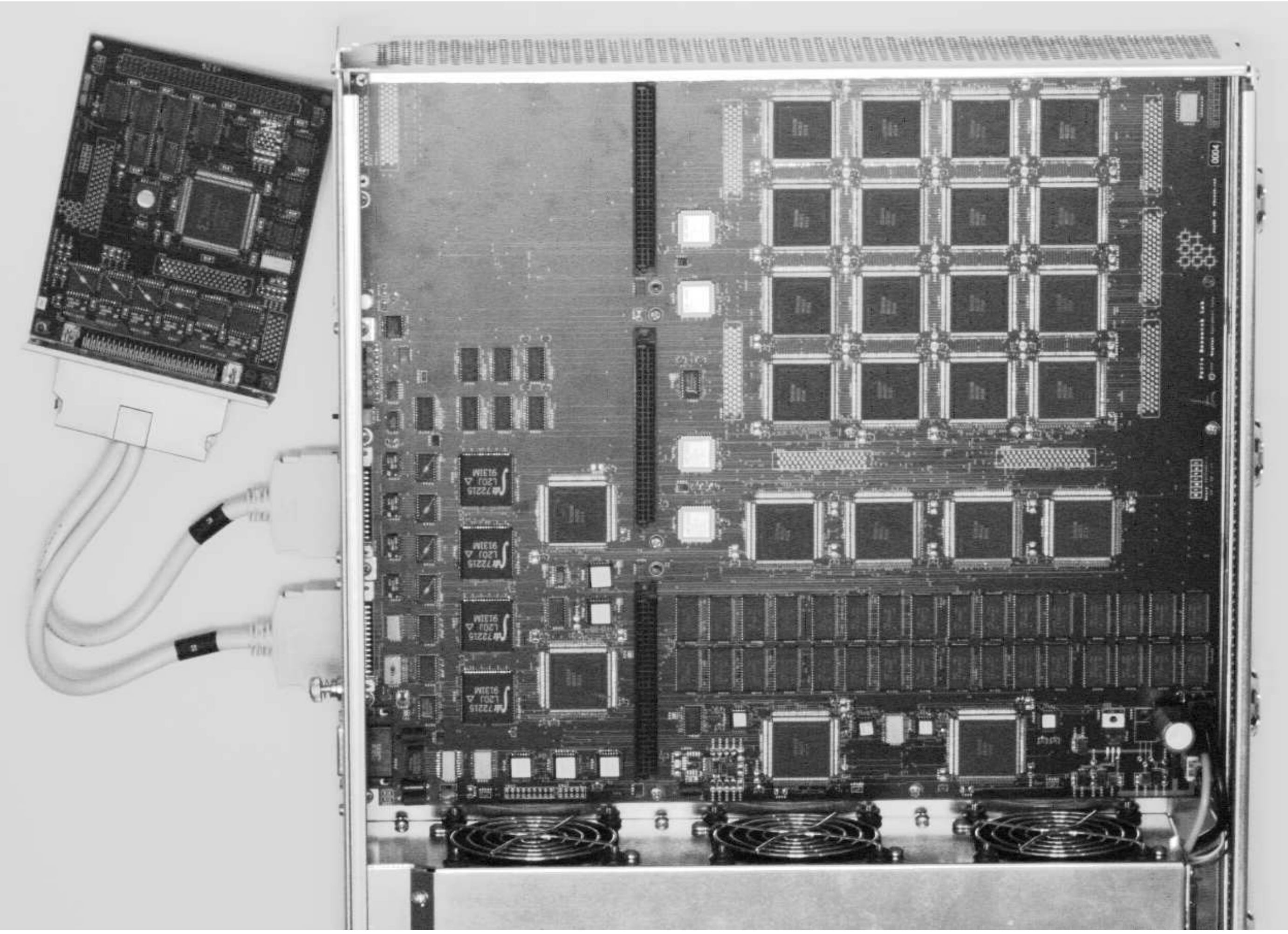


Fig. 4. DECPeRLe-1 and its TURBOchannel interface board

The host connection itself consists of a host-independent part implemented on the P_1 mother board and a host-dependent part implemented on a small option board specific to the host bus. A short cable links the two parts—see figure 4.

In addition to the above, P_1 features daughter-board connectors that can provide more than 1.2 GB/s of bandwidth to specialized hardware extensions.

D. *Firmware*

One extra FPGA on P_1 is not configurable by the user; call it POM, by analogy with ROM. Its function is to provide control over the state of the PAM, through software from the host.

The logical protocol of the host bus itself is *programmed* in POM configuration. Adapting from TURBOchannel to some other logical bus format, such as VME, HIPPI or PCI is just a matter of re-programming the POM and re-designing the small host-dependent interface board.

A function of the POM is to assist the host in downloading a PAM configuration—1.5 Mb for P_1. Thanks to this hardware assist, we are able to reconfigure P_1 up to fifty times per second, a crucial feature in some applications. One can regard P_1 as a *software silicon foundry*, with a 20 ms turn-around time.

We take advantage of an extra feature of the XC3090 component: it is possible to dynamically *read back* the contents of the internal state register of each PAB. Together with a *clock stepping* facility—stop the main clock and trigger clock cycles one at a time from the host—this provides a powerful debugging tool, where one takes a snapshot of the complete internal state of the system after each clock cycle. This feature drastically reduces the need for software simulation of our designs.

PAM designs are synchronous circuits: all registers are updated on each cycle of the same global clock. The maximum speed of a design is directly determined by its *critical combinational path*. This varies from one PAM design to another. It has thus been necessary to design a clock distribution system whose speed can be *programmed* as part of the design configuration. On P_1, the clock can be finely tuned, with increments on the order of 0.01%, for frequencies up to 100 MHz.

A typical P_1 design receives a logically uninterrupted flow of data, through the input FIFO. It performs some processing, and delivers its results, in the same manner, through the output FIFO. The host is responsible for filling-in and emptying-out the other side of both FIFOs. Our firmware supports a mode in which the application clock automatically *stops* when P_1 attempts to read an empty FIFO or write a full one, effectively providing fully automatic and transparent *flow-control*.

The full firmware functionality may be controlled through host software. Most of it is also available to the hardware design: all relevant wires are brought to the two controller FPGAs of P_1. This allows a design to synchronize itself, in the same manner, with some of the external links. Another unique possibility is the *dynamic tuning of the clock*. This feature is used in designs where a slow and infrequent operation—say changing the value of some global controls every 256 cycles—coexists with fast and frequent operations. The strategy is then to slow the clock down before the infrequent operation—every 256 cycles—and speed it up afterwards—for 255 cycles. Tricky, but doable.

E. Other Reconfigurable Systems

Besides our PAMs, which were built first at INRIA in 1987 up to Perle-0, whose architecture is described in some detail in an earlier report [9], then at DEC-PRL, other successful implementations of reconfigurable systems have been reported, in particular at the universities of Edinburgh [10] and Zurich [11], and at the Supercomputer Research Center in Maryland [12].

The ENABLE machine is a system, built from FPGAs and SRAM, specifically constructed at the university of Mannheim [13] for solving the TRT problem of section V-G.2. Many similar application-specific machines have been built in the recent years: their reconfigurable nature is exploited only while developing and debugging the application. Once complete, the final configuration is frozen, once and for all—until the next "hardware release".

Commercial products already exist: QuickTurn [14] sells large configurable systems, dedicated to hardware emulation. Compugen [15] sells a modular PAM-like hardware, together with several configurations focusing on genetic matching algorithms. More systems exist than just the ones mentioned here.

A thorough presentation of the issues involved in PAM design, with alternative implementation choices, is given by Bertin [16].

IV. PAM programming

A PAM program consists of three parts:

1. The *driving software*, which runs on the host and controls the PAM hardware.
2. The logic equations describing the synchronous hardware implemented on the PAM board.
3. The placement and routing directives that guide the implementation of the logic equations onto the PAM board.

The driving software is written in C or C++ and is linked to a runtime library encapsulating a device driver. The logic equations and the placement and routing directives are generated algorithmically by a C++ program. As a deliberate choice of methodology, all PAM design circuits are *digital* and *synchronous*. Asynchronous features—such as RAM write pulses, FIFO flags decoding or clock tuning—are pushed into the firmware (POM) where they get implemented once and for all.

A full P_1 design is a large piece of hardware: excluding the RAM, twenty-three XC3090 containing 15k PABs are roughly the equivalent of 200k gates. This amount of logic would barely fit in the largest gate arrays available in 1994.

The goal of a P_1 designer is to encode, through a 1.5 Mb bitstream, the logic equations, the placement and the routing of fifteen thousand PABs in order to meet the performance requirements of a compute-intensive task. To achieve this goal with a reasonable degree of efficiency, a designer needs full control over the final logic implementation and layout. In 1992, no existing *computer-aided design* (CAD) tool was adapted to such needs.

Emerging synthesis tools were too wasteful in circuit area and delay. One has to keep in mind that we already pay a performance penalty by using SRAM-based FPGAs instead of raw silicon. Complex designs can be synthesized, placed and routed automatically only when they do not attempt to reach high device utilization; even then, the resulting circuitry is significantly slower than what can be achieved by careful hand placement.

Careful low-level circuit implementation has always been possible through a painful and laborious process: schematic capture. For PAM programming, schematic capture is not a viable alternative: it can provide the best performance, but it is too labor intensive for large designs.

Given these constraints, we have but one choice: a middle-ground approach where designs are described algorithmically at the structural level, and the structure can be annotated with geometry and routing information to help generate the final physical design.

A. Programming tools

We first had to choose a programming language to describe circuits. Three choices were possible: a general-purpose programming language such as C++, a hardware description language such as VHDL, or our own language. We do not discuss the latter approach here; it is the subject of current research.

We decided to use C++ for reasons of economy and simplicity. VHDL is a complex, expensive language. C++ programming environments are considerably cheaper, and we are tapping a much wider market in terms of training, documentation and programming tools. Though we had to develop a generic software library to handle netlist generation and simulation, the amount of work remains limited. Moreover, we keep full control over the generated netlist, and we can include circuit geometry information as desired.

A.1 The netlist library

To describe synchronous circuits with our C++ library is straightforward. We introduce a new type `Net`, overload the boolean operators to describe combinational logic, and add a primitive for the synchronous register. From these, a C++ program can be written which *generates* a netlist representing any synchronous circuit. This type of low-level description is made convenient by the use of basic programming techniques such as arrays, `for` loops, procedures and data abstraction. Figure 5 shows, for example, a piece of code representing a generic n-bit ripple-carry adder.

```
template<int N>
struct RippleAdder: Block {
  RippleAdder(): Block("RippleAdder"){}
  void logic(Net<N>& a, Net<N>& b, Net<N>& c,
             Net<N>& sum, Net& carry) {
    input(a); input(b); input(c);
    output(sum); output(carry);
    for (int i = 0; i < N; i++){
      sum[i] = a[i] ^ b[i] ^ c[i];
      carry[i] = (a[i] & b[i])
               | (b[i] & c[i])
               | (c[i] & a[i]);
    }
  }
};
```

Fig. 5. Circuit description in C++

The execution of such a program builds a netlist in memory; this netlist can be analyzed and translated into an appropriate format (XNF or EDIF), or used directly for simulation. Linking a netlist description program with behavioral code yields mixed-mode simulation with no special effort.

Since we have direct access to the netlist at this level of description, we can easily annotate logic operators with placement directives. For example, to specify that our ripple-carry adder should be aligned vertically, with the paired carry and sum bits generated by the same logic block, we simply add the lines shown in figure 6 to the description of the adder.

```
  void placement(Net<N>& sum, Net<N>& carry) {
     for (int i = 0; i < N; i++) {
      carry[i] <<= sum[i];
      sum[i+1] <<= sum[i] + OFFSET(0,1);
  }
```

Fig. 6. Circuit layout in C++

Contrary to the silicon compilers from a decade ago [17], these placement annotations do not affect the logic behavior of the generated netlist. They do not specify contacts; they only specify the partitioning of logic into physical blocks and the absolute or relative placement of these blocks in a two-dimensional grid. A back-end tool analyzes these attributes and emulates the interface of a schematic capture software in order to guarantee that the placement and logic partitioning information is preserved by the FPGA vendor software.

A.2 The runtime library

At the system level, the programming environment provides two main functions: a device driver interface, and full simulation support of that interface. This simulation capability allows the designer to operate together the hardware and software parts from a PAM program. The device driver interface provides the mandatory controls to the application program: the usual UNIX I/O interface with open, close, synchronous and asynchronous read and write; download of the configuration bitstreams for the PAM FPGAs; readback of their state (i.e. the values of all PAB registers); read and write of the PAM static RAMs; software control of the PAM board clock.

A.3 Lessons

The main lesson we draw from our experience with these programming tools is that PAM programming is much easier than ASIC development. Students with no electrical engineering background were able to use our tools after a few weeks of training. In particular, users can easily develop their own module generators in matters of days, while only highly skilled engineers are able to write module generators for custom VLSI. This capability is one of the main reasons why we were able to develop such complex applications spanning dozens of chips, with engineers and students not previously exposed to PAM, each in a matter of months.

B. *Debugging and optimization tools*

Debugging of a PAM design can be done entirely through software. Mixed-mode simulation at the block level allows designers to certify datapath components before using them in complex designs. Full-system simulation eliminates the need for generating special input patterns to test the hardware part of the program. Full-system simulation allows for *hardware/software codebug*: both application driver and hardware, working together.

After simple bugs have been removed, it becomes necessary to simulate the design on a large number of cycles. To do so, the most effective technique is to compile the design into a bitstream, download this bitstream into the board, and run the board in *trace* mode (single-step the clock, readback the board state at each cycle and collect these states for analysis; it is possible to run this mode at up to 100 Hz). In simple cases, this can be done with no modification to the runtime application source code. In complex cases, all necessary primitives are available to build application-specific code to generate and/or analyze the readback traces.

P_1's clock generator can also be operated in *double-step* mode. In that mode, the clock runs at full speed every second cycle. By comparing double-step traces taken at increasing clock frequencies with a previously recorded single-step trace, we can automatically locate the critical path of a design for a given execution. This method alleviates the need to rely on delay simulation as provided by the standard industrial simulation packages. It is necessary to perform that tedious task only once, when certifying the operating speed of the final design.

We developed a screen visualization tool called `showRB` to help analyze readback traces. It can display the state of every flip-flop in every FPGA of a PAM board, at the rate of tens of frames per second. In conjunction with the double-step mode, it can be used to detect critical paths along execution traces. Interestingly enough, such a tool also proved invaluable in *demonstrating* the structure of some hardware algorithms.

V. Applications

Our applications were chosen to span a wide range of current leading-edge computational challenges. In each case, we provide a brief description of the design, a performance comparison with similar reported work, and pointers to publications describing the work in more detail.

One paradigm was systematically applied:

> *Cast the inner loop in PAM hardware; let software handle the rest!*

In what follows, $a \div b$ denotes the quotient and $a \cdot\!|\!\cdot b$ the remainder in the euclidean integer division of a by b.

A. Long integer arithmetic

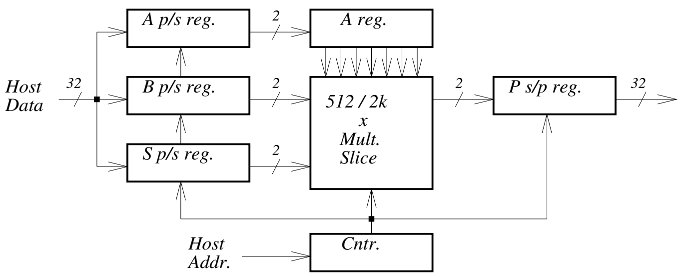


Fig. 7. Long multiplication

PAMs may be configured as long integer multipliers [18]. They compute the product

$$P = A \times B + S$$

where A is an n-bit long multiplier, and B, S are arbitrary size multiplicands and summands [19]; n may be up to 2k for the P_1 implementation.

Our multipliers are interfaced with the public domain arbitrary-precision arithmetic package *BigNum* [20]: programs based on that software automatically benefit from the PAM, by simply linking with an appropriatedly modified BigNum library.

P_1 computes product bits at 66 Mb/s (using radix 4 operations at 33 MHz), which is faster than *all* previously published benchmarks. This is 16 times over the figures reported by Buell and Ward [21] for the Cray II and Cyber 170/750. P_1's multiplier can compute a 50-coefficient 16-bit polynomial convolution (FIR filter) at 16 times *audio real time* (2×24-bit samples at 48 kHz).

The first operational version of this multiplier was developed in less than a week. Two subsequent versions, which refined the design on the basis of actual performance measurements, were each developed in less than 5 man-days.

A more aggressive multiplier design is reported by Louie and Ercegovac [22]: using radix 16 and deep pipeline, this multiplier operates at 79 MHz, which is 2.5 faster than ours within 3 times the area. At that speed, this design is faster than conventional multipliers, even for short 32-bit operands.

B. RSA cryptography

To investigate further the tradeoffs in our hybrid hardware and software system, we have focused on the RSA cryptosystem [23]. Both encryption and decryption involve computing modular exponentials, which can be decomposed as sequences of long modular multiplications, with operand sizes ranging from 256 bits to 1k bits.

Starting from the general-purpose multiplier above, we have implemented a series of systems spanning two orders of magnitude in performance, over three years.

Our first system [18] uses three differently programmed Perle-0 boards, all operating in parallel with the host. At 200 kb/s decoding speed, this was faster than *all* existing 512-bit RSA implementations, regardless of technology, in 1990. A survey by E. Brickell [24] grants the previous speed record for 512-bit key RSA decryption to an ASIC from AT&T, at 19 kb/s.

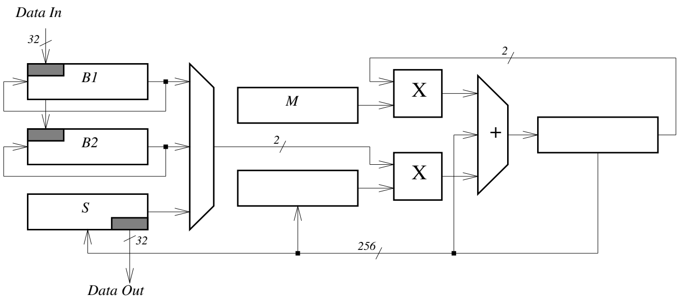


Fig. 8. RSA cryptography

Table I recalls the various original hardware algorithms used in our latest implementation of RSA, and quantifies each speedup achieved.

TABLE I
RSA speedup techniques

Algorithm	Speedup
Chinese remainders	4
Precompute powers	1.25
Hensel's division	1.5
Carry completion	≈ 2
Quotient pipelining	4

The resulting P_1 design for RSA cryptography combines all of the techniques above (see Shand and Vuillemin [25] for details). To fully exploit the available logic gates the P_1 design operates with 970-bit keys. At this key length it delivers an RSA secret decryption rate of 185 kb/s. This is an order of magnitude faster than any previously reported running implementation. For 512-bit keys the same datapath delivers a decryption rate in excess of 300 kb/s although it uses only half the logic resources in P_1.

PAM implementations of RSA rely on reconfigurability in many ways: we use a different PAM design for RSA encryption and decryption; we generate a different hardware modular multiplier for each different prime modulus with the coefficients of the binary representation of each modulus hardwired into the logic equations of the design.

C. Molecular biology

Given an alphabet $\mathcal{A} = (a_1, \ldots, a_n)$, a probability $(S_{ij})_{i,j=1\ldots n}$ of substitution of a_i by a_j, and a probability $(I_i)_{i=1\ldots n}$ (resp. $(D_i)_{i=1\ldots n}$) of insertion (resp. deletion) of a_i, one can use a classical dynamic programming algorithm to compute the probability of transforming a word w_1 over $\mathcal{A}$ into another one w_2; this defines a *distance* between words in $\mathcal{A}$.

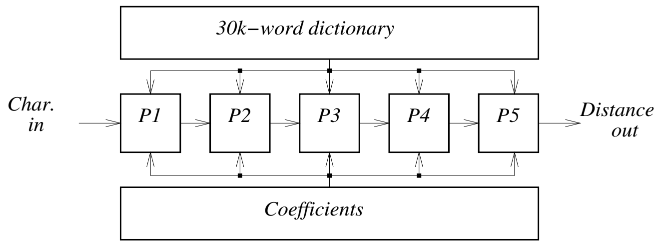


Fig. 9. String matching

Applications include automated mail sorting through OCR scanners, on-the-fly keyboard spelling corrections, and DNA sequencing in biology.

D. Lavenier from IRISA (Rennes, France) has implemented this algorithm with a Perle-0 design which computes the distance between an input word and all 30k words in a dictionary; it reports the k words found in the dictionary which are closest to the input. The system processes 200k words/s which is faster than a solution previously implemented at CNET using 12 Transputers. It has only half of the performance obtained by a system previously developed at IRISA based on 28 custom VLSI chips and two printed-circuit boards.

The DNA matching algorithm [26] is one of the driving applications for the PAM developed at the Supercomputing Research Center in Maryland [12]: the reported performance is, here again, in excess of that obtained with existing supercomputers.

The *Compugen* commercial company [15] sells the *Bioccelerator*, a PAM which can be configured as a number of molecular biology search functions. This device is a co-processor to a host server; it can be accessed through remote procedure call from any workstation on the network. It is interfaced with a widely used software package and its use is transparent, except for the speed-up advantages.

D. Heat and Laplace equations

Solving the heat and Laplace equations has numerous applications in mechanics, integrated circuit technology, fluid dynamics, electrostatics, optics and finance [27].

The classical *finite difference method* [28] provides computational solutions to the heat and Laplace equations. Vuillemin [29] shows how to speed-up this computation with help from special-purpose hardware. A first implementation of the method on P_1, by Vuillemin and Rocheteau [29], operates with a pipeline depth of 128 operators. Each operator computes $\frac{T+T'}{2}$, where T and T' are 24-bit temperatures.

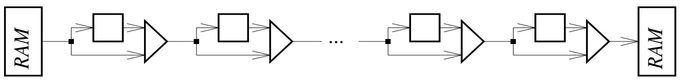


Fig. 10. Heat and Laplace equations

At 20 MHz, this first design processes 5G operations—add and shift—per second. For such a smooth problem, one can easily show [29] that fixed-point yields the same results as floating-point operations. The performance achieved by this first 24-bit P_1 design thus exceeds those reported by McBryan *et al.* [30] [31], for solving the same problems with the help of supercomputers. A sequential computer must execute 20 billion instructions per second in order to reproduce the same computation.

S. Hadinger and P. Raynaud-Richard further improved the implementation [32]. Refining the statistical analysis, they show that the datapath width can be reduced to 16 bits provided the rounding-off of the low-order bit is done *randomly*—with all deterministic round-off schemes, parasitic stable solutions exist which significantly perturb the result. Their implementation therefore uses a 64-bit linear feedback shift-register to randomly set the rounding direction for each processing stage.

The width reduction in the datapath allows us to extend the pipeline length to 256, pushing the equivalent processing power up to 39 GIPS. Using P_1's fast DMA-based I/O capabilities and a large buffer of host memory, this design can accurately simulate the evolution of temperature over time in a 3-D volume, discretized on 512^3 points, with arbitrary power source distributions on the boundaries. It also supports the use of *multigrid* simulation, where one "zooms out" to coarser discretization grids in order to rapidly advance in simulated time, then "zooms back in" to full resolution, in order to accurately smooth out the desired final result.

E. Neural networks

M. Skubiszewski [33] [34] has implemented a hardware emulator for binary neural networks, based on the *Boltzmann machine* model.

The Boltzmann machine is a probabilistic algorithm which minimizes quadratic forms over binary variables, i.e.

expressions of the form

$$E(\vec{N}) = \sum_{i=0}^{n-1} \sum_{j=0}^{i} w_{i,j} N_i N_j$$

where $\vec{N} = (N_0, \ldots, N_{n-1})$ is a vector of binary variables and $(w_{i,j})_{0 \le i,j < n}$ is a fixed matrix of *weights*. It is typically used to find approximate solutions to some $\mathcal{NP}$-hard problems, such as graph partitioning and circuit placement.

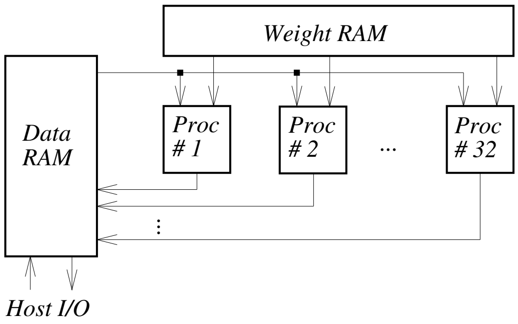


Fig. 11. Boltzmann machine

The latest P_1 realization solves problems with 1400 binary variables, using 16-bit weights, for a total computing power of 500 *megasynapses per second*. (The megasynapse is the traditional unit used in this field; it amounts to one million additions and multiplications by small coefficients.)

F. Multi-standard video compression

In view of the required input bandwidth (30 MB/s for standard TV color images) and the amount of computation required by current standards (resp. 3, 4 and 8 Gop/s[1] for JPEG, DCT^{3D} and MPEG), custom hardware is currently necessary for compressing video in real time.

Matters get complicated, as several different video compression standards are emerging. The following shows how a single configurable system such as P_1 can perform, through different designs, three (or more) of the current leading standards.

JPEG The computation specified by the *Joint Photographic Expert Group* is performed in three stages, according to:

$$\begin{matrix}\textit{Source}\\ \textit{Image}\end{matrix} \mapsto \boxed{\mathcal{R}} \mapsto \boxed{\mathcal{DCT}^2} \mapsto \boxed{\mathcal{Q}} \mapsto \boxed{\mathcal{A}/\mathcal{HC}} \mapsto \begin{matrix}\textit{Compressed}\\ \textit{Image}\end{matrix}$$

The initial RAM $\mathcal{R}$ is used to store 8 consecutive lines in the input image, with double buffering. It feeds the $\mathcal{DCT}^2$ module with 8×8 square sub-images.

1. The two-dimensional $\mathcal{DCT}^2$ (Discrete Cosine Transform) maps 8×8 squares from the space to the frequency domain.
2. Each frequency coefficient is divided by a number $Q = Q_{x,y}$. The choice of the *quantization* table Q provides a way to control the compromise between the compression factor and the quality of the decompressed image.
3. Run-length, and arithmetic or Huffman encodings $\mathcal{A}/\mathcal{HC}$ are performed on the quantized values.

MPEG The *Motion Picture Expert Group* system does *motion compensation* ($\mathcal{MC}$) by computing a correlation between blocks within two time-consecutive images. The result is difference-coded, then goes through a processing similar to JPEG.

$$\begin{matrix}\textit{Digital}\\ \textit{Video}\end{matrix} \mapsto \boxed{\mathcal{R}} \mapsto \boxed{\mathcal{MC}} \mapsto \boxed{\Delta\,\mathit{Code}} \mapsto \begin{matrix}\textit{Compressed}\\ \textit{Video}\end{matrix}$$

MPEG-1 requires storage for only 4 images, after allowing for double buffering. The decoder is much simpler than the encoder, however, the MPEG decoder still requires about as much hardware as the following DCT^{3D}.

A detailed FPGA mapping of the motion estimation algorithm—the core of the MPEG standard—is given by Furtek [35]. Mapping this fully laid-out design onto P_1 would be a straightforward task.

DCT^{3D} J. Vuillemin, D. Martineau, and J. Barraquand from PRL have used P_1 to experiment with DCT^{3D}, a 3-D version of JPEG—the third dimension being time. Except for RAM, this method requires only half as much hardware as MPEG. It leads to an excellent compression factor, with an appropriate choice of the quantization table $Q = Q(x, y, t)$, a 512-entry cube. Early experiments indicate that, for a given compression rate, the quality of the restituted video is (subjectively) better with DCT^{3D}than with MPEG.

The method performs the following sequence of computations:

$$\begin{matrix}\textit{Digital}\\ \textit{Video}\end{matrix} \overset{30}{\mapsto} \boxed{\mathcal{R}} \overset{30}{\mapsto} \boxed{\mathcal{DCT}^3} \overset{60}{\mapsto} \boxed{\mathcal{A}/\mathcal{HC}} \overset{\approx 2}{\mapsto} \begin{matrix}\textit{Compressed}\\ \textit{Video}\end{matrix}$$

In this diagram, the numbers on the arrows indicate the transfer bandwidth, in MB/s.

- The algorithm needs a video buffer big enough to store 8 consecutive images (twice for double buffering). Thus, DCT^{3D}requires 4 times more RAM than MPEG.
- Past the initial video buffer, all the processing is performed in a straight pipeline operating on *video cubes* of size $8^3 \times 16$b, made of eight 8×8 squares consecutive in time.

This P_1 design computes 48 fixed-point operations (32-bit outputs add, subtract, multiply and shift) at 25 MHz, for a total of 1.4G operations per second. Based on our specification software, we rate this algorithm, which requires a lot of data movement, at 15 GIPS.

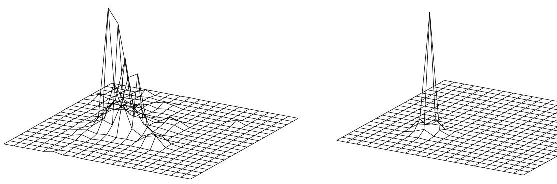
[1] 10^9 16-bit integer operations per second

G. High-energy physics

G.1 Image classification

The *calorimeter* is part of a series of benchmarks proposed by CERN[2] [36]. The goal is to measure the performance of various computer architectures, in order to build the electronics required for the *Large Hadron Collider* (LHC), before the turn of the millennium. The calorimeter is challenging, and well documented: CERN benchmarks seven different electronic boxes, including some of the fastest current computers, with architectures as different as DSP-based multiprocessors, systolic arrays and massively parallel systems.

This problem is typical of high-energy physics data acquisition and filtering: $20 \times 20 \times 32$b images are input every 10 μs from the particle detectors, and one must discriminate within a few μs whether the image is interesting or not. This is achieved by computing some simple statistics on it (maximum value, second-order moment,...) and using them to decide whether or not a sharp peak is present (figure 12). What makes the problem difficult here are the high input bandwidth (160 MB/s) and the low latency constraint.



Hadron jet Electron

Fig. 12. Calorimeter typical input images

Vuillemin [7] analyzes in detail the possible implementations of the calorimeter, on both general-purpose computer architectures (single and multi processors, SIMD and MIMD) and special-purpose electronics (full-custom, gate-array, FPGAs). The conclusion provides an accurate quantitative analysis of the computing power required for this task: the PAM is the only structure found to meet this bound.

This algorithm was implemented by P. Boucard and J. Vuillemin on P_1 [37] [38]. Using the external I/O capabilities described in section III-C, data is input from the detectors through two off-the-shelf HIPPI-to-TURBOchannel interface boards plugged directly onto P_1. The datapath itself uses about half of P_1's logic and RAM resources, for a virtual computing power of 39 GBOPS (figure 13).

G.2 Image analysis

The *Transition Radiation Tracker* (TRT) is another benchmark from CERN, analyzed in the same report [36]. The problem is to find straight lines (particule trajectories) in a noisy digital black and white image.

The algorithm used is based on the classical *Hough transform*: first compute the number of active ("on") pixels on each possible line crossing the image (here the physics of the problem limits the candidate lines to those having a small positive or negative slope), then select the line which has the maximum number of active pixels, or discard the image if no line has a sufficient number of active pixels. As above, the rate of the input data (160 MB/s) and the low latency requirement (≤ 2 images) preclude any implementation solution other than one using specialized hardware, as shown by CERN [36].

R. Männer and his team from University of Mannheim [13] have successfully built the specialized FPGA-based ENABLE machine for solving this problem, using the straightforward $O(N^3)$ implementation of the Hough transform. It computes the score for all lines of 16 different slopes crossing a 128×96 grid at the required 100 kHz rate, with a latency of 2 images (20 μs). It needs more than twice the computing power of P_1 to achieve this result.

J. Vuillemin [39] describes an $O(N^2 \log N)$ algorithm to compute the Hough transform, in a recursive way analogous to the Fast Fourier Transform (figure 14). The resulting gain in the processing power needed by the computation makes it just possible to fit it in one P_1 board.

This was implemented by L. Moll, P. Boucard and J. Vuillemin [37] [38]. As above, data is directly input from the detectors through two HIPPI-to-TURBOchannel boards plugged in P_1's extension slots. The design computes 31 slopes at the required 100 kHz rate with a latency of 1 image (10 μs). A 64-bit sequential processor would need to run at 1.2 GHz to achieve the same computation.

G.3 Cluster detection

The NESTOR Neutrino Telescope under construction in the Mediterranean near Pylos, Greece, is an three-dimensional array of 168 photomultiplier tubes (PMTs) designed to detect Cherenkov radiation from fast muons created by neutrino interactions. Clustered detections from actual Cherenkov-generated photons are expected to happen at a maximum rate of a few per second, while the background noise originating from bioluminescence and radioactive potassium (^{40}K) causes random PMT firings at a rate of 100 kHz per PMT.

A P_1 board will be used to process the raw data and detect muon trajectories[3], by looking for space- and time-correlation among events. The peak and average data rates are 500 MB/s and 100 MB/s respectively. Data enters directly through P_1's 256b-wide daughter-board connectors (see section III-C). Provided the peak data rate can be accommodated—which is the case with the P_1 solution—subsequent processing is straightforward (see Katsanevas *et al.* [40] for details).

H. Image acquisition

P_1's TURBOchannel adapter (see section III-C), being built around a single XC3090, is a PAM in its own right—albeit a small one. M. Shand [41] describes a number of

[2] European Organization for Nuclear Research, Geneva, Switzerland

[3] In high-energy physics terminology, this is the *first level trigger*.

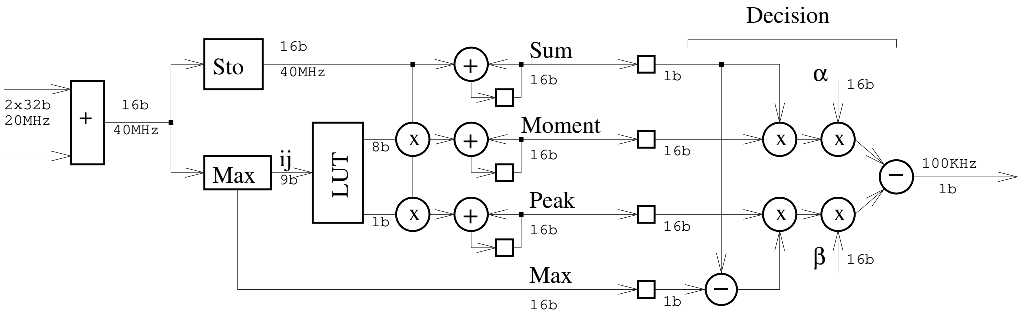


Fig. 13. Calorimeter datapath

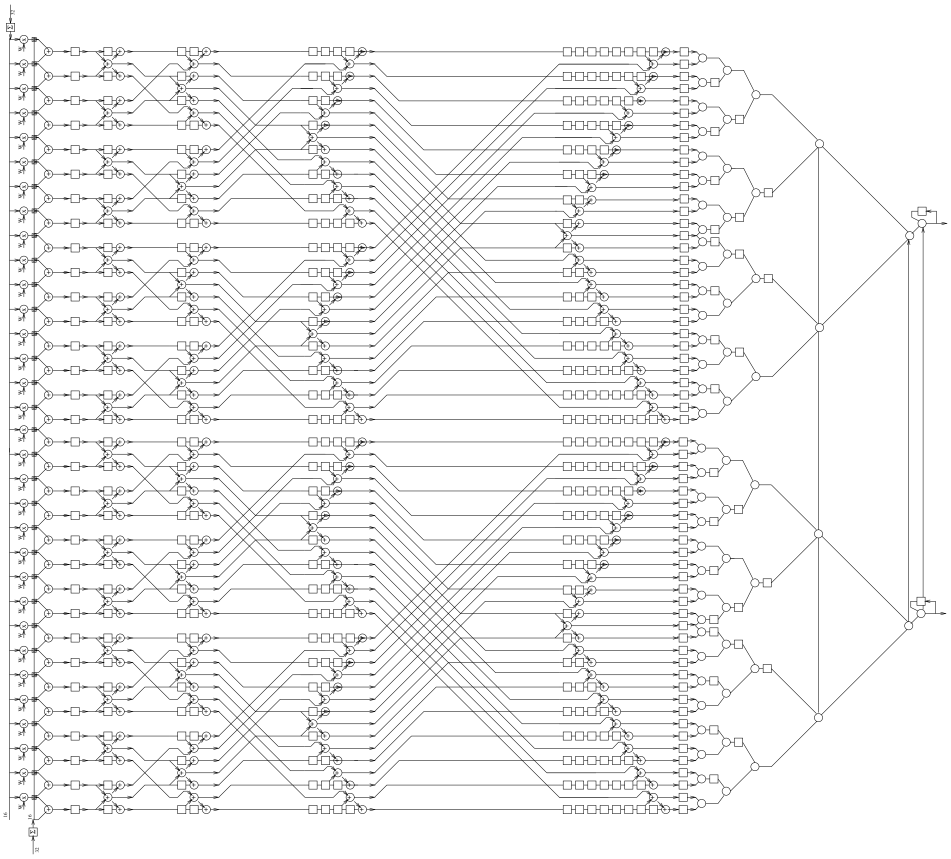


Fig. 14. Fast Hough transform

experiments based on this board, including an interface to a large frame CCD camera [42]. This camera delivers image data at 10 MB/s with no flow control. Conventionally an interface for such a camera would use a dedicated frame buffer. Our interface dispenses with this buffer by transfering the incoming image data directly into system memory, using Direct Memory Access (DMA) over the TURBOchannel. In addition to the obvious cost savings of eliminating the frame buffer memory, use of system memory makes the captured image immediately available to software and allows the system to capture images continuously. These attributes prove essential to one use of this interface—the principal image acquisition system at the Swedish Vacuum Solar Telescope where the system has been in use since May 1993 [43].

The success of this small PAM (or *PAMette*) has lead us to develop a new PAM board, I/O-oriented and of small size, to explore these new kinds of applications. M. Shand, in collaboration with G. Scharmer and Wang Wei of the Swedish Royal Observatory, is investigating the use of this board in an adaptive optics system combining image acquisition, image processing, and on-the-fly servo control.

I. Stereo vision

Part of the research on stereo vision at INRIA[4] is focused on computing dense, accurate and reliable *range maps*, from simultaneous images obtained by two cam-

[4]Institut de Recherche en Informatique et Automatique, Sophia-Antipolis, France.

eras. The selected stereo matching algorithm is presented by Faugeras *et al.* [44]: a recursive implementation of the score computation makes the method independent of the size of the correlation window, and the calibration method does not require the use of a calibration pattern.

Stereo matching is integrated in the navigation system of the INRIA cart, and used to correct for inertial and odometric navigation errors. Another application, jointly with CNES[5], uses stereo to construct digital elevation maps for a future planetary rover.

A *software* implementation of the selected method computes the correlation between a pair of images in 59 seconds on a SPARCStation II. A dedicated hardware implementation using four digital signal processors (DSP), developed jointly by INRIA and Matra MSII, performs the same task in 9.6 seconds. A P_1 implementation of the very same algorithm by L. Moll [45] runs over thirty times faster, in 0.28 seconds: a key step towards real-time stereo matching.

This design uses the full 100 MB/s bandwidth available between P_1 and its host. It also relies on fast reconfiguration, as the processing is a straight pipeline between three distinct PAM configurations, which are successively swapped in time for each image pair processed.

J. Sound synthesis

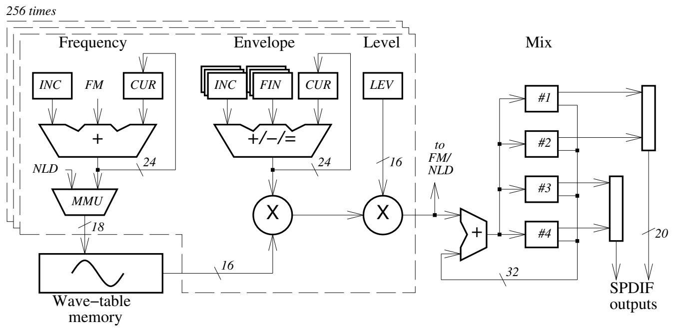


Fig. 15. Sound synthesizer

In order to explore the digital signal processing domain, D. Roncin and P. Boucard implemented a real-time digital audio synthesizer on P_1, capable of producing up to 256 independant voices at a sampling rate of 44.1 kHz. Primarily designed for the use of additive synthesis techniques based on lookup-tables, this implementation includes features which allow frequency-modulation synthesis and/or non-linear distortion and can also be used as a sampling machine.

This design contains 4 MB of wave-table memory, shared by the 256 voice generators, which can be partitioned into sub-tables of various sizes allowing the simultaneous use of up to 1k different sound patterns. It also includes an ouput mixing section and global control.

Each of the 256 voices consists of:

- A phase computation section, which computes the index of a voice sample in the selected wave-table (using 24-bit arithmetic). Using the output of another voice in this computation leads to frequency modulation and non-linear distortion.
- An envelope generator and static level section, which computes the amplitude value for the current sample (also using 24-bit arithmetic) and combines it with the output of the wave-table to produce the amplitude modulated sample. Dynamic amplitude envelopes are generated using linked linear segment techniques.
- A control section, which defines the operating mode of the voice: normal oscillator, carrier operator for frequency modulation, non-linear transfer function operator, free-running or single shot, synchronous phase operation, wave-table size and location selection, output channel selection...

The output mixing section contains four 32-bit accumulators, which connect to two SPDIF[6] (stereo) digital audio output ports. Synthesizing this standard consumer audio format allows for the *direct* connection of P_1 to an off-the-shelf tape recorder or audio amplifier, through a mere cable.

All parameters and controls can be updated by the host at any time in parallel with the running synthesis. At 22 MHz, this design produces 11M samples per second, which amounts to about 22M 16×16-bit multiplications, 100M ALU operations and 45M load/store operations. A software implementation of this algorithm running on standard CPUs shows that the DECPeRLe-1 implementation is equivalent to a computing power of about 2 GIPS. A simpler version of this design has been ported on a standard DSP processor (27-MHz Motorola 56001). The DSP is capable of computing only 24 voices at the required sampling rate—less than one tenth the number computed by P_1.

K. Long Viterbi Decoder

In many of today's digital communications systems the signal-to-noise ratio (SNR) of the link has become the most severe limitation. Convolutional encoding with maximum likelihood (Viterbi) decoding provides a means to improve the SNR of a link without increasing the power budget, and has become an important technique in satellite and deep-space communications systems.[7]

The coding gain of a Viterbi system is primarily determined by the *constraint length* K of the code, while the complexity of the decoder increases exponentially with K. Today's VLSI implementations typically offer codes with $K = 7$ and $K = 8$. NASA's Galileo space probe is equipped with a constraint length 15 rate 1/4 encoder, for which a Viterbi decoder based on an array of 256 custom VLSI chips is being developed [46].

R. Keaney and D. Skellern from Macquarie University (Sydney, Australia), together with M. Shand and

[5] Centre National d'Etudes Spatiales, France

[6] Sony/Philips Digital Audio Interface

[7] The same techniques apply to high-density magnetic storage devices, for equivalent reasons.

J. Vuillemin from PRL, have implemented a Viterbi decoder for the Galileo code on P_1 [47]. Using on-board RAM to trace through the 2^{14} possible states of the encoder, this design computes 4 states in parallel at each 40 ns clock cycle, for an overall decoding speed of 2 kb/s. The coding gain has been measured to be within 0.5 dB of the optimal gain for this particular code.

There is no analytical method to prove that a particular code provides the optimal coding gain for a given constraint length. Taking further advantage of PAM reconfigurability, this system will be used to perform a *code search* among constraint length 15 convolution codes, by recompiling a new P_1 configuration on-the-fly for each code.

VI. The computing power of PAM

Let us now *quantify* the computing power of a PAM processor. Following earlier reports [48] [7], we define the *virtual computing power* of a PAM with n PABs which operate at f Hertz as the product $P = n \times f$. The resulting power P is expressed in *boolean operations per second* (BOPS). For $n = 800$ and $f = 25$ MHz, we find $P = 16$ GBOPS for a leading edge single FPGA in 1992, and 5000 GBOPS = 5 TBOPS in 2001. At 25 MHz, the PAM P_1 has a virtual computing power of 368 GBOPS—roughly equivalent to what we should get in a single FPGA near year 1996.

Our particular choice of unit for measuring computing power is based on the 4-input combinational function[8]. A *bit-serial binary adder*, which is composed of two functions of three inputs, also counts for one unit. The accounting rules which follow, for arithmetic and logic operations over n-bit wide inputs, are thus straightforward:

$\boxed{+}$ One $(n+n \mapsto n+1)$-bit addition each nanosecond is worth n GBOPS. Subtraction, integer comparison and logical operations are bit-wise equivalent to addition.

$\boxed{\times}$ One $(n \times m \mapsto n+m)$-bit multiplication each nanosecond is worth nm GBOPS. Division, integer shifts and transitive (see Vuillemin [49]) bit permutations are bit-wise equivalent to multiplication.

Due to the great variety of the operations required by each application, quantitative performance comparison between different computer architectures is a challenging art [50]. The *million of instructions per second* (MIPS) and *million of floating-point operations per second* (MFLOPS) are more traditional units for measuring computing power. By our definition, a 32-bit standard microprocessor[9] operating at 100 MHz (100 MIPS) has a virtual computing power of 3.2 GBOPS, and a 200 MHz, 64-bit processor features 12.8 GBOPS. A 100-MHz, 64-bit floating-point multiplier delivering one operation per cycle (100 MFLOPS) would rate 281 GBOPS.

It follows from this accounting that P_1 has a virtual computing power which is higher than that of the fastest integer microprocessor existing in 1994.

[8] The particular choice of the unit function only affects our measure by a constant factor, provided we keep bounded fan-in.
[9] with no hardware multiplier

	small	medium	large	
I/O bandwidth	200	400	1k	MB/s
Computing power	50	200	1k	GBOPS
FPGA area	1	4	20	kPABs
RAM size	8	32	160	MB
Unit cost	800	3k	12k	$

TABLE II
Vital figures of currently feasible PAMs

VII. Conclusion

We have shown that it is now possible to build high-performance PAMs, with applications in a large number of domains. Table II updates what is feasible within 1994 technology. The technology curves for PAM cost/performance derive from those for FPGA and static RAM [51]; we can use them as a basis for extrapolation, from now into the future.

Let us compare the respective merits of three possible implementation technologies, for a given specific high-performance system. High-performance means here that the computational requirement far exceeds the possibilities of the fastest micro-processor. That leaves three implementation possibilities: 1– program a parallel machine; 2– design a specific PAM configuration; 3– build a custom system. The first two only involve software; the third involves hardware as well. Let us review some of the comparative merits, for each technology.

1. Each reported PAM design was implemented and tested within one to three months, starting from the delivery of the specification software. This is roughly equivalent to the time it takes to implement a *highly optimized* software version of the same system on a supercomputer: both are technically challenging, yet both are orders of magnitude faster than what it takes to cast a system into custom ASICs and printed-circuit boards.
2. For many specific high-speed computational problems, PAM technology has now proved superior, both in performance *and* cost, to all current forms of *general-purpose* processing systems: pipelined machines, massively parallel ones, networks of microprocessors,...
   The cost of P_1 is comparable to that of a high-end workstation. This is *much* lower than the cost of a supercomputer. Based on figures from McBryan [30], the price (in $ per operation per second) for solving the heat and Laplace equations is 100 times higher with supercomputers than with P_1.
3. PAM technology is currently best applied to low-level, massively repetitive tasks such as image or signal processing. Due to their software complexity, many current supercomputer applications still remain outside the possibilities of current PAM technology.
4. For many *real-time* problems, PAMs already have performance *and* cost equal to those of specific, custom

systems: the lower the volume, the better for the PAM. By tuning a specific application for a PAM, we have shown that *very high performance* implementations are possible. For at least six of the cases presented in section V, the performance achieved by our P_1 implementation exceeds, by at least one order of magnitude, those of any other implementation, including custom VLSI based ones.

5. An important field of applications is accessible only through PAM technology: high-bandwidth interfaces to the external world, with a *fully programmable, real-time* capability. P_1 has 256-bit wide connectors, capable to deliver up to 1.2 GB/s of external I/O bandwidth. It is then a "simple matter of hardware programming" to interface directly to any electrically-compatible external device, by programming its communication protocol into the PAM itself. Applications include high-bandwidth networks, audio and video input or output devices, and data acquisition.

References

[1] W. S. Carter, K. Duong, R. H. Freeman, H. C. Hsieh, J. Y. Ja, J. E. Mahoney, L. T. Ngo and S. L. Sze, "A user programmable reconfigurable logic array", *IEEE 1986 Custom Integrated Circuits Conference*, pp. 233–235, 1986.

[2] Xilinx, Inc., *The Programmable Gate Array Data Book*, Xilinx, 2100 Logic Drive, San Jose, CA 95124, USA, 1993.

[3] D. D. Hill, B. K. Britton, B. Oswald, N. S. Woo, S. Singh, T. Poon and B. Krambeck, "A new architecture for high-performance FPGAs", *Field Programmable Gate Arrays: Architecture and Tools for Rapid Prototyping*, H. Gruenbacher and R. W. Hartenstein, editors, Lecture Notes in Computer Science Nr. 705, Springer-Verlag, 1993.

[4] Algotronix Ltd., *The Configurable Logic Data Book*, Edinburgh, UK, 1990.

[5] Concurrent Logic, Inc., *Cli6000 Series Field-Programmable Gate Arrays*, Concurrent Logic Inc., 1270 Oakmead Parkway, Sunnyvale, CA 94086, USA, 1992.

[6] GEC Plessey Semiconductors, *ERA60100 Electrically Reconfigurable Array Data Sheet*, GEC Plessey Semiconductors Ltd., Swindon, Wiltshire SN2 2QW, UK, 1991.

[7] J. E. Vuillemin, "On computing power", *Programming Languages and System Architectures*, J. Gutknecht, editor, Lecture Notes in Computer Science Nr. 782, Springer-Verlag, pp. 69–86, 1994.

[8] Digital Equipment Corp., *TURBOchannel Hardware Specification*, DEC document EK-369AA-OD-007B, 1991.

[9] P. Bertin, D. Roncin and J. Vuillemin, "Introduction to Programmable Active Memories", *Systolic Array Processors*, J. McCanny, J. McWhirter and E. Swartzlander Jr., editors, Prentice-Hall, pp. 301–309, 1989.

[10] T. Kean and I. Buchanan, "The use of FPGAs in a novel computing subsystem", *1st International ACM Workshop on Field-Programmable Gate Arrays*, pp. 60–66, Berkeley, CA, USA, 1992.

[11] B. Heeb and C. Pfister, "Chameleon, a workstation of a different color", *Field Programmable Gate Arrays: Architecture and Tools for Rapid Prototyping*, H. Gruenbacher and R. W. Hartenstein, editors, Lecture Notes in Computer Science Nr. 705, Springer-Verlag, 1993.

[12] J. Arnold, D. Buell and E. Davis, "Splash II", *4th ACM Symposium on Parallel Algorithms and Architectures*, San Diego, CA, USA, pp. 316–322, 1992.

[13] F. Klefenz, K. H. Noffz, R. Zoz and R. Maenner, "ENABLE—A systolic 2nd-level trigger processor for track finding and e/pi discrimination for ATLAS/LHC", *Proc. IEEE Nucl. Sci. Symp.*, San Francisco, CA, USA, pp. 62–64, 1993.

[14] Quickturn Systems, Inc., *RPM Emulation System Data Sheet*, Quickturn Systems, Inc., 325 East Middlefield Road, Mountain View, CA 94043, USA, 1991.

[15] Compugen, *The Bioccelerator*, product brief, Compugen Ltd., 10 Hayetsira St., Rosh-Ha'ayin, 40800 Israël, 1993.

[16] P. Bertin, *Mémoires actives programmables: conception, réalisation et programmation*, Thèse de Doctorat, Université Paris 7, 75005 Paris, France, 1993.

[17] D. D. Gajski, editor, *Silicon Compilation*, Addison-Wesley, 1988.

[18] M. Shand, P. Bertin and J. Vuillemin, "Hardware speedups in long integer multiplication", *Computer Architecture News*, vol. 19(1), pp. 106–114, 1991.

[19] R. F. Lyon, "Two's complement pipeline multipliers", *IEEE Trans. on Comm.*, vol. COM-24, pp. 418–425, 1976.

[20] B. Serpette, J. Vuillemin and J. C. Hervé, *BigNum: A Portable Efficient Package for Arbitrary-Precision Arithmetic*, PRL report 2, Digital Equipment Corp., Paris Research Laboratory, 85, Av. Victor-Hugo, 92563 Rueil-Malmaison Cedex, France, 1989.

[21] D. A. Buell and R. L. Ward, "A multiprecise integer arithmetic package", *The Journal of Supercomputing*, vol. 3, pp. 89–107, Kluwer Academic Publishers, Boston, MA, USA, 1989.

[22] M. E. Louie and M. D. Ercegovac, "A variable precision multiplier for field-programmable gate arrays", *2nd International ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, Berkeley, CA, USA, February 1994.

[23] R. L. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *CACM*, vol. 21(2), pp. 120–126, 1978.

[24] E. F. Brickell, "A survey of hardware implementations of RSA", *Crypto '89*, Lecture Notes in Computer Science Nr. 435, Springer-Verlag, pp. 368–370, 1990.

[25] M. Shand and J. Vuillemin, "Fast implementation of RSA cryptography", *11th IEEE Symposium on Computer Arithmetic*, Windsor, Ontario, Canada, pp. 252–259, 1993.

[26] D. P. Lopresti, "P-NAC: a systolic array for comparing nucleic acid sequences", *Computer*, vol. 20(7), pp. 98–99, 1987.

[27] R. P. Feynman, R. B. Leighton and M. Sands, *The Feynman Lectures on Physics*, 3 volumes, Addison-Wesley, 1963.

[28] R. Dautray and J. L. Lions, *Mathematical Analysis and Numerical Methods for Sciences and Technology*, 9 volumes, Springer-Verlag, 1990.

[29] J. E. Vuillemin, "Contribution à la résolution numérique des équations de Laplace et de la chaleur", *Mathematical Modelling and Numerical Analysis*, edited by AFCET Gauthier-Villars, RAIRO, vol. 27(5), pp. 591–611, 1993.

[30] O. A. McBryan, "Connection Machine application performance", *Scientific Applications of the Connection Machine*, World Scientific, pp. 94–114, 1989.

[31] O. A. McBryan, P. O. Frederickson, J. Linden, A. Schüller, K. Solchenbach, K. Stüben, C-A. Thole and U. Trottenberg, "Multigrid methods on parallel computers—a survey of recent developments", *Impact of Computing in Science and Engineering*, vol. 3(1), pp. 1–75, Academic Press, 1991.

[32] S. Hadinger and P. Raynaud-Richard, *Résolution numérique des équations de Laplace et de la chaleur*, rapport d'option, Ecole Polytechnique, 91128 Palaiseau Cedex, France, 1993.

[33] M. Skubiszewski, "A hardware emulator for binary neural networks", *1990 International Neural Network Conference*, vol. 2, pp. 555–558, Paris, 1990.

[34] M. Skubiszewski, "An exact hardware implementation of the Boltzmann machine", *1992 International Conference on Application-Specific Array Processors*, Dallas, TX, USA, 1992.

[35] F. Furtek, "A field-programmable gate array for systolic computing", *1993 Symposium on Integrated Systems*, pp. 183–200, The MIT press, Cambridge, MA, USA, 1993.

[36] J. Badier, R. K. Bock, Ph. Busson, S. Centro, C. Charlot, E. W. Davis, E. Denes, A. Gheorghe, F. Klefenz, W. Krischer, I. Legrand. W. Lourens, P. Malecki, R. Männer, Z. Natkaniec, P. Ni, K. H. Noffz, G. Odor, D. Pascoli, R. Zoz. A. Sobala, A. Taal, N. Tchamov, A. Thielmann, J. Vermeulen and G. Vesztergombi, "Evaluating parallel architectures for two real-time applications with 100 kHz repetition rate", *IEEE Trans. Nucl. Sci.*, vol. 40(1), pp. 45–55, 1993.

[37] D. Belosloudtsev, P. Bertin, R. K. Bock, P. Boucard, V. Dörsing, P. Kammel, S. Khabarov, F. Klefenz, W. Krischer, A. Kugel, L. Lundheim, R. Männer, L. Moll, K. H. Noffz, A. Reinsch, M. Shand, J. Vuillemin and R. Zoz, "Programmable Active Memories in real-time tasks: implementing data-driven triggers for LHC experiments", to appear in the *Journal of Nuclear Instruments and Methods for Physics Research*, Elsevier Publishers, Amsterdam, NL, 1995.

[38] L. Moll, J. Vuillemin and P. Boucard, "High-energy physics on DECPeRLe-1 Programmable Active Memory", to appear in

ACM International Symposium on FPGAs, Monterey, CA, USA, February 1995.

[39] J. E. Vuillemin, "Fast linear Hough transform", *1994 International Conference on Application-Specific Array Processors*, pp. 1–9, IEEE Computer Society Press, 1994.

[40] S. Katsanevas, M. Shand and J. Vuillemin, "DECPeRLe-1 implementation of NESTOR's first level trigger", *3rd NESTOR International Workshop*, Pylos, Greece, October 1993.

[41] M. Shand, *Measuring System Performance with Reprogrammable Hardware*, PRL report 19, Digital Equipment Corp., Paris Research Laboratory, 85, Av. Victor-Hugo, 92563 Rueil-Malmaison Cedex, France, August 1992.

[42] Kodak Motion Analysis Systems, *Kodak Megaplus Camera, Model 1.4*, Eastman Kodak Company, March 1992.

[43] G. W. Simon, P. N. Brandt, L. J. November, G. B. Scharmer and R. A. Shine, "Large-scale photospheric motions: first results from an extraordinary eleven-hour granulation observation", *Solar Surface Magnetism*, R. J. Rutten and C. J. Schrijver, editors, NATO ASI Series C433, Kluwer, 1994.

[44] O. Faugeras, T. Viéville, E. Théron, J. Vuillemin, B. Hotz, Z. Zhang, L. Moll, P. Bertin, H. Mathieu, P. Fua, G. Berry and C. Proy, *Real Time Correlation-Based Stereo: Algorithm, Implementations and Applications*, research report 2013, INRIA, 06902 Sophia-Antipolis, France, 1993.

[45] L. Moll, *Implantation d'un algorithme de stéréovision par corrélation sur mémoire active programmable PeRLe-1*, rapport de stage, Ecole des Mines de Paris, Centre de Mathématiques Appliquées, 06904 Sophia-Antipolis, France, 1993.

[46] J. Statman, G. Zimmerman, F. Pollara and O. Collins, "A long constraint length VLSI Viterbi decoder for the DSN", *TDA Progress Report 42-95*, Jet Propulsion Laboratory, Pasadena, CA, USA, July-Sept. 1988.

[47] R. A. Keaney, L. H. C. Lee, D. J. Skellern, J. E. Vuillemin and M. Shand, "Implementation of long constraint length Viterbi decoders using Programmable Active Memories", *11th Australian Microelectronics*, Surfers Paradise, QLD Australia, 1993.

[48] P. Bertin, D. Roncin and J. Vuillemin, "Programmable Active Memories: a performance assessment", *Symposium on Integrated Systems*, Seattle, WA, USA, MIT Press, 1993.

[49] J. E. Vuillemin, "A combinatorial limit to the computing power of VLSI circuits", *IEEE Transactions on Computers*, April 1983.

[50] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 1990.

[51] C. P. Thacker, *Computing in 2001*, Digital Equipment Corporation, Systems Research Center, 130 Lytton, Palo Alto CA94301, USA, 1993.

Jean E. Vuillemin is a graduate from Ecole Polytechnique. He received a Ph. D. from Stanford University in 1972, and one from Paris University in 1974.
He taught Computer Science at the University of California, Berkeley in 1975, and Université d'Orsay from 1976 to 1980. He was at INRIA from 1980 to 1987, and at DEC-PRL from 1988 to 1994. He is now professor at Faculté Léonard de Vinci.
He has authored over 100 papers on program semantics, algorithm design and analysis, combinatorics and hardware structures.
His current research interests concern programmable hardware, theory, implementations and applications.

Patrice Bertin received the Engineer degree from Ecole Polytechnique (Palaiseau, France) in 1984, and the Ph. D. in Computer Science degree from Université Paris 7 (Paris, France) in 1993. From 1988 to 1994, he worked on the PAM project at Digital Equipment Corporation's Paris Research Laboratory, as a visiting scientist from INRIA (Institut National de Recherches en Informatique et en Automatique, Rocquencourt, France). He is currently with the new Léonard-de-Vinci University in La Défense near Paris, France. His E-Mail address is `<Patrice.Bertin@inria.fr>`.

Didier Roncin received degrees in Electrical Engineering, Computer Science, Musicology and Computer Music from Paris University. He worked at IRCAM (Paris, France) on research for acoustic and computer music from 1977 to 1984. He joined Jean Vuillemin's team at INRIA from 1984 to 1987 where they started the PAM project in 1987. He went to Digital Equipment Corporation's Paris Research Laboratory from 1977 to 1994, where he worked principally on the PAM project's hardware architectures. He is currently at the Léonard-de-Vinci University in Paris, France, where he is investigating designs of generic PCI-based and low cost PAMs, as well as specific PAM architectures for digital audio and computer music applications. His E-Mail address is `<Didier.Roncin@inria.fr>`.

Mark Shand attended the University of Sydney where he received BS degree in 1981 and PhD degree in 1987. His thesis was on VLSI CAD. He spent 1987 and 1988 with the Australian Government's CSIRO continuing his VLSI CAD work. He was employed at Digital Equipment Corporation's Paris Research Laboratory from 1989 to 1994 where he worked principally on the Programmable Active Memories project. He is currently at Digital Equipment Corporation's Systems Research Center in Palo Alto, California. His E-Mail address is `<shand@acm.org>`.

Hervé H. Touati received the Ph. D. degree from U.C. Berkeley in 1990. From 1991 to 1994 he was a research scientist at Digital Equipment Corporation's Paris Research Laboratory. He co-founded Xorix SARL.

Philippe Boucard received the Engineer degree from Ecole Nationale Supérieure des Télécommunications (Paris, France) in 1981. From 1991 to 1994, he worked on the PAM project at Digital Equipment Corporation's Paris Research Laboratory. He is currently with Matra MHS (France), in the microcontroller design department. His E-Mail address is `<Philippe.Boucard@matramhs.fr>`.