Programmable Active Memories: a Performance Assessment

Patrice Bertin, Didier Roncin
and Jean Vuillemin
Digital Equipment Corporation
Paris Research Laboratory
85, avenue Victor-Hugo
92563 Rueil Malmaison Cedex, France

Abstract

We present some quantitative performance measurements for the computing power of Programmable Active Memories (PAM), as introduced by [2]. Based on Field Programmable Gate Array (FPGA) technology, the PAM is a universal hardware co-processor closely coupled to a standard host computer. The PAM can speed up many critical software applications running on the host, by executing part of the computations through a specific hardware design. The performance measurements presented are based on two PAM architectures and ten specific applications, drawn from arithmetics, algebra, geometry, physics, biology, audio and video. Each of these PAM designs proves as fast as any reported hardware or super-computer for the corresponding application. In cases where we could bring some genuine algorithmic innovation into the design process, the PAM has proved an order of magnitude faster than any previously existing system (see [19] and [18]).

1 PAM concept

Like any RAM memory module, a PAM is attached to the system bus of a host computer. The processor can write into, and read from the PAM. Being an active hardware co-processor however, the PAM processes data between write and read instructions. The specific processing is determined by the content of its *configuration memory*. The host can change the PAM configuration by *downloading* a new design into it, within a few milliseconds.

We speed up a specific software application running on the host, by executing its *critical inner-loop* through an appropriate hardware design downloaded into the PAM. Ten examples of such designs and applications are presented below.

In selecting them, we have attempted to cover as many application areas as we could. In each, we picked basic and frequent problems, where a large inner-loop speedup through a specific PAM design results in a significant speedup of the application system, software and hardware combined.

For the sake of debugging, the functionality of each hardware design is matched, bit-wise at the I/O level, by a software implementation. Through successive refinements, both hardware and software implementations have been optimized for speed, while retaining the I/O behavior. This is our experimental data in this assessment of PAM computing power.

The traditional measurement unit here is the Mips[1], or its multiple the Gips ($1G = 10^9$) which is more appropriate to the levels of performance reported here. As explained in [9], quantitative performance comparisons between different computer architectures is a challenging art. For one, our Mips were measured in different units: for the 32b MIPS R3000 or the 64b Alpha AXP instruction sets? for which clock speed? cache size? bus bandwidth? For second, in a number of cases we had external data for both VLSI and super-computer implementations of systems close enough to some of ours that we could envision to make reliable speed comparisons.

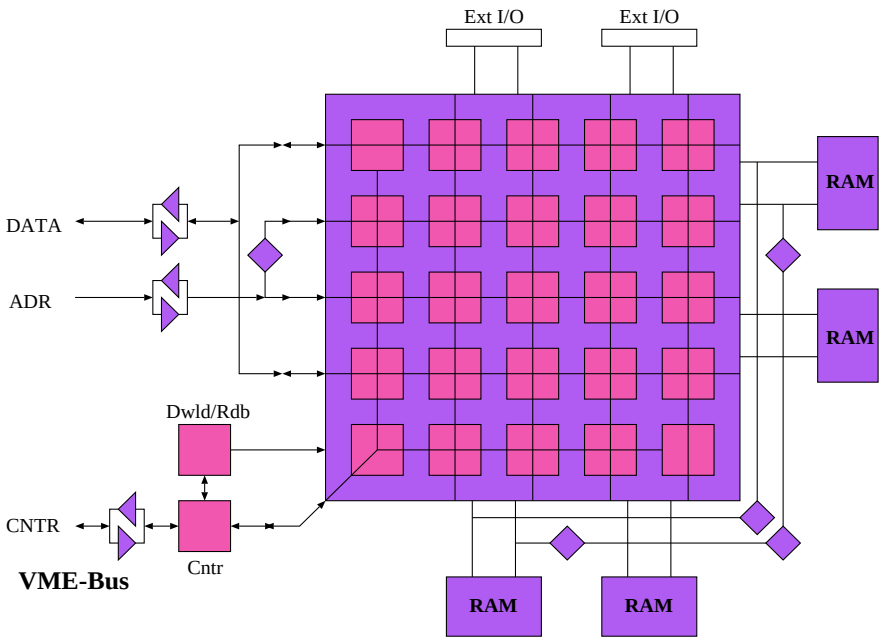
To make quantitative comparisons significant across such a wide spectrum of feasible implementation technologies, we introduce a common unit for measuring all forms of computing power, the *Gbops* (billion of binary operations per second) where each useful boolean operation with up to 4 inputs, be it clocked or not, counts for one. For example the computing power of a 32b adder at 100 Mhz is 6.4 Gbops, accounting for both carries and sums. That of a n bits multiplier at 1 GHz is $2n^2$ Gbops. Any specific computer operation can be similarly decomposed at the bit level, so we can evaluate the corresponding power in Gbops.

2 Two PAM architectures

Our assessment is based on two PAM architectures realized at DEC's Paris Research Lab., DECPeRLe-0 (see [2]) and DECPeRLe-1 (see [3]) which we respectively refer to as P_0 and P_1 in the following.

Each PAM is built around a large array of bit-level configurable logic cells (hereafter called Programmable Active Bits or PABs) in which the application-specific hardware operator is programmed. This array is surrounded with local RAM banks used as a cache (wide and fast enough to match the PAM's external bandwidth), a programmable clock generator, and some additional non-configurable logic to manage the host bus interface and the download process.

[1] Million of instructions per second



The above figure sketches the architecture of P_0 (1988). The central computational array is made of a 5×5 matrix of Xilinx XC3020 Programmable Gate Arrays [23]; it has two 32-bit wide RAM banks on the south and east sides, a VME bus interface on the west side and general-purpose interface connectors on the north side. The control and bootstrap logic is implemented in two extra XC3020 (non user-programmable). Finally, additional bus switching resources are provided for global data routing (represented here with diamond-shape boxes).

The architecture of P_1 was designed after two years of P_0 usage and is described in details in [3]. It features a 4-times-bigger central computational matrix with accordingly wider RAM, a faster host interface, and a much more flexible global interconnection network for data routing and switching.

For the purpose of evaluating Xilinx-based designs such as ours, it is convenient to define the Programmable Active Bit as in [2]: a PAB consists of a universal 4 inputs combinatorial gate and a synchronous flip-flop. Using this measurement unit, our two PAM architectures P_0 and P_1 have the following vital statistics:

PAM	part	nb	PABs	$F_{\max}$ (MHz)	Power (Gbops)	RAM (MB)	Host bus (MB/s)
P_0	XC3020	25	3.2K	25	80	0.5	8
P_1	XC3090	23	14K	40	588	4	100

This chart exhibits the three most important architectural parameters conditioning which application benefits most from a PAM speed-up:

- The number of PABs (3.2K for P_0 and 14K for P_1), together with the application-dependent maximal clock frequency (25 MHz and 40 MHz)

at which we can reliably operate. The product of these two numbers is the maximum theoretical computing power of the PAM, expressed in Gbops (80 Gbops for P_0 and 588 Gbops for P_1).

- The host bus bandwidth: 8 MB/s for P_0 through a VME bus, and 100 MB/s for P_1 through a TURBOchannel interface [6].
- The size of the local (fast) RAM: 0.5 MB for P_0 and 4 MB for P_1.

3 PAM programming

Programming an algorithm on the PAM is similar to casting it in conventional hardware (gate-array or VLSI), with two very important differences however: first the target hardware provides a completely clean implementation of the synchronous logic model, so there is no need to worry about low-level electrical details; second the whole design process is purely software with a fast (5 to 30 mn) turnaround, so it can be approached with the same methodology of piecewise testing and successive refinements as a software design. For a given application, it involves:

1. identifying the critical computations which are best implemented in hardware; this is usually done by successive refinement, under constraints of communication bandwidth and load balancing between the software and hardware;
2. implementing the hardware part on the PAM and gradually optimizing it;
3. implementing the software part on the host processor and gradually optimizing it.

Step 3 above is done with conventional techniques. Step 2 consists in describing the logic design to implement in the PAM down to the individual bit level, as well as its geometric layout, much as is done for a conventional VLSI design.

For this, we have developed a tool suite in which the hardware design is described by the writing of a program in a conventional programming language (we use Lisp, C++ [21] and Esterel [1]) using a specialized library. This program will describe the various logic modules by their bit-level logic equations, or by using standard library modules (adders, registers, standard interfaces...). It will also contain layout (geometric) information to the level of details the designer wishes to specify: this usually starts with global floorplanning and detailed layout of the important datapath components only, to be enriched with more precise descriptions along the optimization process.

Executing this program produces a partially placed, hierarchical netlist which can either be simulated, or compiled to the final PAM configuration in

a fully automatic way, through tools developed in-house (board-level routing, logic optimization) and standard Xilinx back-end software (chip-level placement and routing, bitstream generation).

The design can then be downloaded into the PAM for debugging and testing under real conditions; its maximum running speed can be characterized, and its critical paths identified (we have developed specialized interactive tools to help visualize the latter). The designer can then gradually optimize it, for example by adding extra levels of pipeline to increase the clock speed, or optimizing the geometric layout, or adding new functionalities to further unload the software. The most important point there is that, as compiling and trying a new version costs no money and only a few minutes to an hour of time, it is possible to incrementally design, test and optimize the algorithms and implementations involved, as is usually done with software, as opposed to having to have them correct and optimal on the first try.

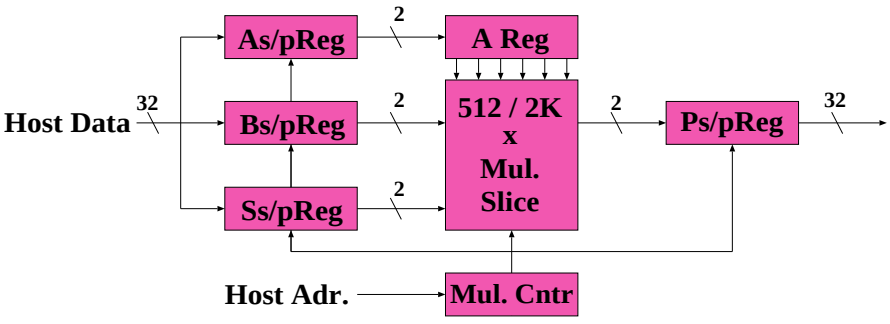
4 Ten PAM applications

The following 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, the names of the implementors, and a performance comparison with similar reported work. In each case, the following simple paradigm has been applied:

> *compile the inner loop in PAM hardware, and let software handle the rest!*

In what follows, we let $(a \div b)$ represent the quotient and $(a \cdot|\cdot b)$ the remainder in the integer division.

4.1 Long multiplication



We have programmed both PAMs into long multipliers ($n = 512$ bits for P_0, and $n = 2K$ bits for P_1) computing $P = A \times B + S$, with A a n-bits multiplier, and B, S arbitrary size multiplicands and summands (see [10], [2]

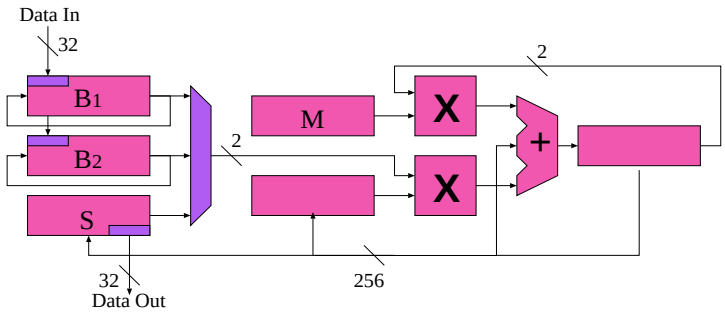
and [19]). These multipliers are interfaced with an arbitrary-precision arithmetic package *BigNum* (see [16]) so that any program based on that software takes advantage of the PAM without modification, by simply relinking with a modified BigNum library. This respectively speeds up raw multiplication by a factor up to 24 (P_0) and 30 (P_1) for long operands, as compared to optimized assembly code running on the host workstation. For example, P_0 equipped with this design computes RSA encryption/decryption at 1500 bits per second for arbitrary 512-bit keys (see [14]); this is about 10 times faster than our best software version on the same host.

The P_1 implementation produces product bits at 66 Mbits/s, which makes it faster than *any* known machine for which we could obtain benchmark measures. It is at least 16 times faster than the best figures for a Cray II or a Cyber 170/750 reported by [5]. This multiplier can be used to compute a 50 coefficients (16-bits) polynomial convolution (FIR filter) at 16 times *audio real time* (2×24 bits samples at 48 kHz).

4.2 RSA cryptography

To further investigate the tradeoffs which are possible in our hybrid hardware/software system we focused on the RSA cryptosystem (see [14]), which can be cast entirely in terms of long multiplications. Starting from the above general-purpose multiplier, M. Shand from DEC-PRL implemented a series of hardware/software systems spanning two orders of magnitude in performance. The latest version is based on an original hardware design for computing modular products at the rate of two bits per cycle [20].

The system originally used three differently programmed P_0 boards, all operating in parallel with the host (see [19]). At 200 kbit/s decoding speed, it was faster than *any* currently existing 512 bits RSA implementation, in *any* technology, as of February 1990. A survey by [4] grants the previous speed record for 512 bits keys RSA decryption to a VLSI from AT&T, at 19 kbits/sec.

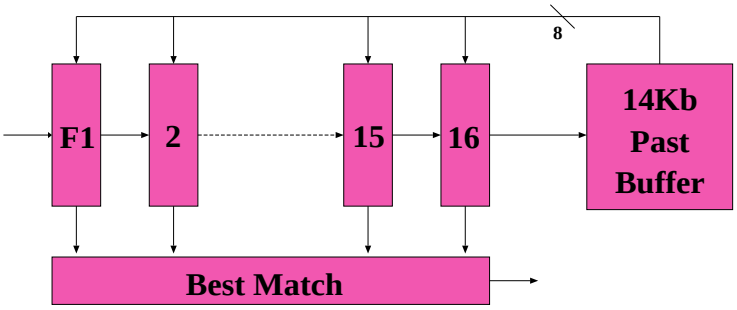


M. Shand recently ported this RSA system to a single P_1 board; at 40 MHz, this design provides either two independent 600 Kb/s RSA encryption channels for 480b keys, or one 175 Kb/s RSA encryption channel for 970b

keys.

4.3 Data compression

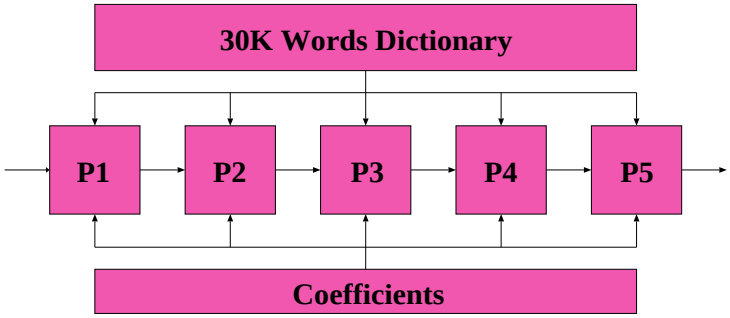
M. Skubiszewski from DEC-PRL has implemented a P_0 design to speed up the algorithm of [24], which is well known to achieve an average data compression ratio varying from 2 for English (or French, or Polish...) plain text to 3 for C (or Lisp, or Pascal...) source code.



The design is a massively parallel method which computes 64 byte comparisons on each (70 ns) cycle; it matches the next 16 bytes in the file to be compressed against the last 4k bytes seen (stored in the local RAM), in order to detect the longest substring previously seen. While this design performs a respectable 1 Gops (8 bits integer comparison), it ends up in a disappointing factor two speed-up, when compared to optimized software such as Unix `compress`. Indeed, such optimized software avoids most of the comparisons performed in the hardware, by detecting early that they are irrelevant to the final output. A more elaborate hardware design is needed to genuinely speed up this particular algorithm.

4.4 String matching

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 a probability of transformation of w_1 into w_2; this defines a *distance* between any two words w_1 and w_2 over $\mathcal{A}$.

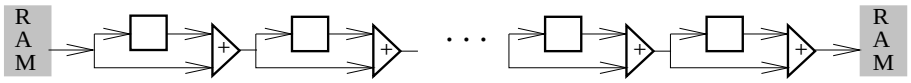


D. Lavenier from IRISA (Rennes, France) has implemented this algorithm with a P_0 design which computes the distance between an input word and all 30K words in a dictionary; it reports the k words found in the dictionary which are closest to the input. The system processes 200K words/sec: this is faster 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 2 PC boards.

Applications of this algorithm include automated mail sorting using OCR scanners, on-the-fly keyboard spelling corrections, and DNA sequence matching (see [11]).

4.5 Heat and Laplace equations

[22] shows how to adapt the classical finite difference method (see [7]) to compute solutions of the heat and Laplace equations in n dimensions with help from special purpose hardware. An implementation of the method on P_1 operates with a pipe-line depth of 128 operators:



Each operator computes:

$$\mathcal{O}(v_0, v_1) = \begin{cases} v_0, \text{ when } v_0 \cdot|\cdot 2 = 1, \\ 2(v_0 \div 4 + v_1 \div 4 + (v_0 \cdot|\cdot 4) \div 2), \text{ otherwise,} \end{cases} \quad (1)$$

all with 24b fixed-point data format. At 20 MHz, this amounts to 5 Gops (24b adds, tests and shifts); it is easy to show (see [22]) that fixed-point gives the same results as floating-point operations for this specific problems; the achieved performance thus exceeds those reported in [12] and [13] for solving the same problem with super-computers. A sequential computer needs to execute 25 billion instructions per second (25 Gips), to reproduce the same computation.

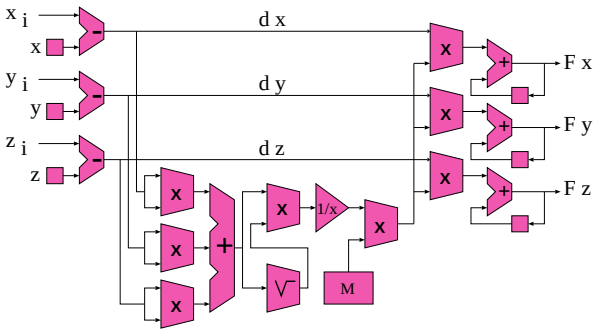
The heat and Laplace equations have many applications in mechanics, circuit technology, fluid dynamics, electrostatics, optics, finance...

4.6 Newton's mechanics

J. Vuillemin has specified a P_1 design for computing the evolution of a n-body system, using Newton's equations. The design computes the gravitational field acting on body k by summing the individual fields induced at k by each other body in the system. This amounts to the following 18 operations:

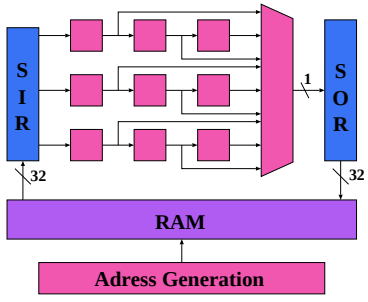
$(x_i - x_k) \Rightarrow dx$	$(dx2 + dz2) \Rightarrow dxz$	$fm \times dx \Rightarrow fdx$
$(y_i - y_k) \Rightarrow dy$	$(dxz + dy2) \Rightarrow d2$	$fm \times dy \Rightarrow fdy$
$(z_i - z_k) \Rightarrow dz$	$\sqrt{d2} \Rightarrow d$	$fm \times dz \Rightarrow fdz$
$(dx \times dx) \Rightarrow dx2$	$d \times d2 \Rightarrow d3$	$fx + fdx \Rightarrow fx$
$(dy \times dy) \Rightarrow dy2$	$\frac{1}{d3} \Rightarrow fd$	$fy + fdy \Rightarrow fy$
$(dz \times dz) \Rightarrow dz2$	$fd \times m_i \Rightarrow fm$	$fz + fdz \Rightarrow fz$

Positions and forces are represented as 20 bits floating-point numbers. Assuming a 40 ns internal cycle (achievable through deep pipe-line) the expected throughput exceeds 2.5 GFlops (this design has not been tested by printing time).



4.7 Binary 2D convolution

B. Chen and J. Vuillemin have implemented a 7×7 binary 2D convolver on P_0, for performing erosion, dilation and matching on black and white images, as defined in [15].



The convolver runs at 25 MHz, generating one pixel each 40 ns; it completes a single convolution pass over one 512×512 image in 10 milliseconds;

this allows for up to 4 successive operations (erosion, dilation, or matching) at video rate. Reproducing this performance through optimized software would require a 200 Mips computer.

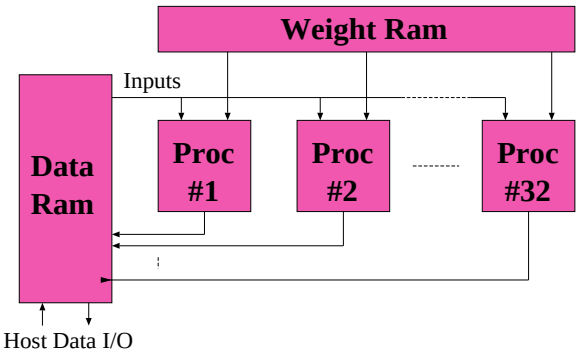
4.8 Boltzmann machine

M. Skubiszewski has implemented two successive versions of a hardware emulator for binary neural networks, based on the *Boltzmann machine* model (see [17] and [18]).

The Boltzmann Machine is a probabilistic algorithm which minimizes quadratic forms over binary variables, i.e. expressions of the form

$$E(\vec{N}) = \sum_{i=0}^{n-1} \sum_{j=0}^{i} w_{i,j} N_i N_j \tag{2}$$

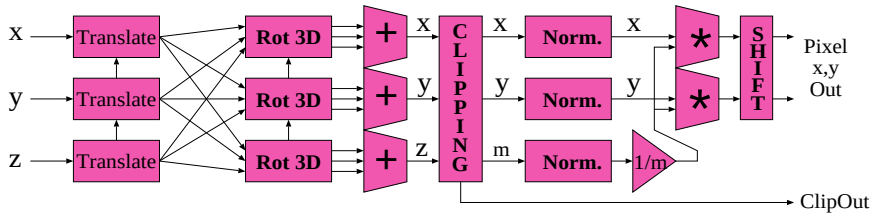
where $\vec{N} = (N_0, \ldots, N_{n-1})$ is a vector of binary variables and $(w_{i,j})_{0 \le i,j < n}$ is a fixed matrix of *weights*. It is typically used to find approximate solutions to $\mathcal{NP}$-hard problems, such as graph partitioning or circuit placement.



The latest realization, on P_1, can solve problems with up to 1400 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, or one million terms of (2)).

4.9 3D Geometry

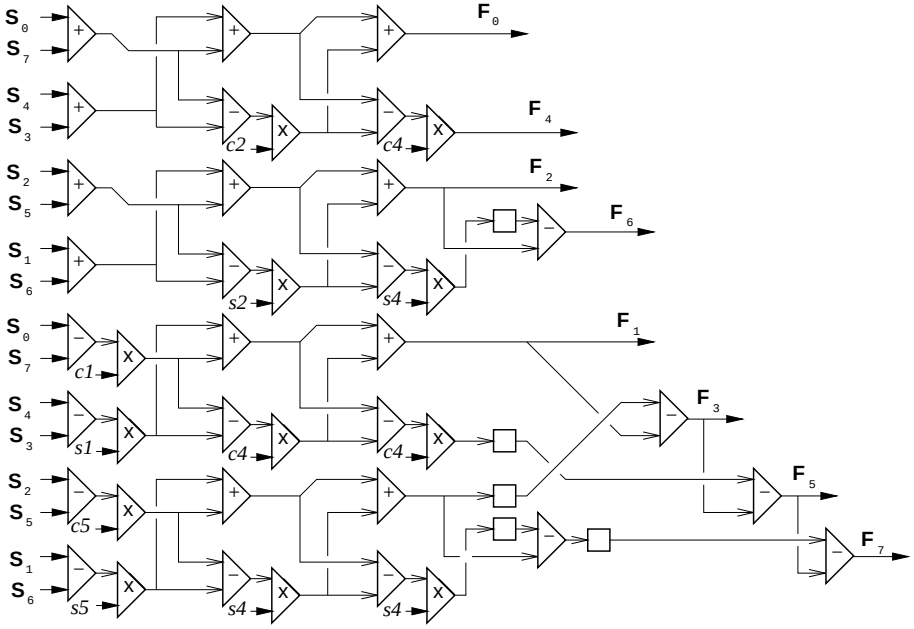
H. Touati from DEC-PRL has implemented a 3-D graphic accelerator for P_1, which supports translation, rotation, clipping and perspective projection, to directly compute the screen image of a cloud of points in 3-D space.



At 25MHz, it has a peak performance of 1.56 million points per second, using 16 bit fixed point coordinates for the input and output, and up to 32 bits for the intermediate results. One needs a 300 Mips processor to achieve the same throughput in software.

4.10 Discrete cosine transform

This design (by J. Vuillemin and D. Martineau on P_1) compresses a video stream in *real time* through multi-dimensional fast discrete cosine transform. The fDCT implements the following network:



The overall design computes 48 fixed-point (32 bits outputs) operations (add, subtract, multiply and shift) on each 40 ns cycle, for a total of 1.4 Gops. To match this performance through software would require a 15 Gips processor.

5 Conclusion

The following chart summarizes the practical PAM performance achieved by each of our ten designs:

Design	PABs	MHz	Gbops	Gops	Gips	PAM
Multiplier	8k	33	264	0.8	2.6	P_1
RSA	8k	32	256	0.5	4	P_1
DCT	10k	25	250	1.4	15	P_1
Newton	10k	25	250	2.5 †	5	P_1
Laplace	10k	20	200	7.5	20	P_1
Boltzmann	8k	25	200	1	1.5	P_1
3D Geometry	3k	25	75	0.5	0.7	P_1
Ziv-Lempel	3k	15	45	1	2	P_0
String	3k	10	30	0.15	0.3	P_0
2D convolution	1k	25	25		1	P_0

†These are Gflops, with 20 bit floating-point numbers.

The applications are ranked according to the most reliable performance measure, namely the *Gbops*. As a comparative measure of resource utilisation in such systems, the following table charts the maximum theoretical performance of generic PAM hardware (in Gbops) obtained by multiplying the maximal clock frequency (in MHz) by the area (in PABs):

PAM	Area	1 MHz	20 MHz	50 MHz
XC3020	128	0.1	2.5	6.4
XC3090	640	0.6	12.8	32
P_0	3.2K	3.2	64	
P_1	14K	14	280	700

Three years of PAM design lead us to believe the following:

1. For each of the chosen application, we have shown that the level of performance achieved with the PAM is comparable to the best figures reported using super-computers or custom silicon circuits.

   Our applications have been carefully selected for having a clearly identified (PAM implementable) inner-loop, which accounts for a vast percentage of the software run-time. For such low level processing, the PAM proves more cost effective than any super-computer.

   Due to their software complexity, many current super-computers applications still remain outside the possibilities of current PAM technology.

2. Each mentioned PAM design was implemented and tested within one or two 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 with a super-computer; both are technically challenging, yet remain an order of magnitude faster than the time it takes to cast a system into silicon.

3. The cost of P_1 is comparable to that of a high-end workstation. This is orders of magnitude lower than the cost of a super-computer. Based on figures from [12], we find that the price (in $ per operation per second) of solving the heat and Laplace equations is 100 times higher with super-computers than with the PAM.

4. Another field of applications, not covered by any existing super-computer, is open to PAM technology: high-bandwidth interfaces to the external world, with *fully programmable real-time* capabilities. The P_1 PAM has a 256b wide connector, capable to deliver up to 6.4 Gb/s of external bandwidth. It is a "simple matter of hardware programming" to interface directly with any electrically-compatible external device, by programming its communication protocol into the PAM itself. Applications for this capability are numerous, including interfaces to high-bandwidth networks, audio and video input or output devices, on-the-fly data acquisition and filtering, etc...

References

[1] Gérard Berry. A Hardware Implementation of Pure Esterel. *PRL Report 15*, Digital Equipment Corp., Paris Research Lab., 85 av. Victor Hugo, 92563 Rueil Malmaison, France, 1991.

[2] Patrice Bertin, Didier Roncin and Jean Vuillemin. Introduction to programmable active memories. *Systolic Array Processors, J. McCanny, J.McWhirther, E. Swartslander Jr. editors*, Prentice Hall, 300–309, 1989. Also available as *PRL report 3*, Digital Equipment Corp., Paris Research Lab., 85, Av Victor Hugo. 92563 Rueil Malmaison Cedex, France, 1989.

[3] Patrice Bertin, Didier Roncin and Jean Vuillemin. Programmable Active memories: the Coming of Age. *PRL report in preparation*, Digital Equipment Corp., Paris Research Lab., 85, Av. Victor Hugo. 92563 Rueil-Malmaison Cedex, France, 1992.

[4] E.F. Brickell. A Survey of Hardware Implementations of RSA. *Proceedings of Crypto'89, Lecture Notes in Computer Science*, Springer Verlag, 1990.

[5] D.A. Buell and R.L. Ward. A multiprecise integer arithmetic package. *The Journal of Supercomputing, Kluwer Academic Publishers*, Boston, 3:89–107, 1989,

[6] Digital Equipment Corporation. TURBOchannel hardware specification. *DEC document EK-369AA-OD-007A*, Digital Equipment Corp., 1991.

[7] R.P. Feynman, R.B. Leighton and M. Sands. The Feynman lectures on PHYSICS. Addison-Wesley, 1963.

[8] J.P. Gray and T. Kean. Configurable hardware: two case studies of micro-grain computation. *Systolic Array Processors, J. McCanny, J.McWhirther, E. Swartslander Jr. editors*, Prentice Hall, 310–319, 1989.

[9] J.L. Hennessy and D.A. Patterson. Computer architecture: a quantitative approach. *Morgan Kaufmann Publishers, Inc*, 1990.

[10] R.F. Lyon. Two's complement pipeline multipliers. *IEEE Trans. Comm.*, COM-24:418–425, 1976.

[11] D.P. Lopresti. P-NAC: A systolic array for comparing nucleic acid sequences. *Computer Magazine*, 20(7):98–99, 1987.

[12] O.A. McBryan. Connection machine application performance. In *Scientific Applications of the Connection Machine, World Scientific*, 94–114, 1989.

[13] O.A. McBryan, P.O. Frederickson, J. Linden, A. Shüller, 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 Engineering, Academic Press*, 3(1):1–75, 1991.

[14] R.L. Rivest, A. Shamir and L. Adleman. Public key cryptography. *CACM 21*, 120–126, 1979.

[15] J.P. Serrat. Image Analysis and Mathematical Morphology. *Academic Press, N.Y.*, 1982.

[16] 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 Lab., 85, av. Victor Hugo. 92563 Rueil Malmaison Cedex, France, 1989.

[17] Marcin Skubiszewski. A hardware emulator for binary neural networks. *Proceedings of the International Neural Network Conference, Paris*, 2:555–558, 1990.

[18] Marcin Skubiszewski. An exact hardware implementation of the Boltzmann Machine. *PRL Report in preparation*, Digital Equipment Corp., Paris Research Lab., 85, av. Victor Hugo. 92563 Rueil Malmaison Cedex, France, 1992.

[19] Mark Shand, Patrice Bertin and Jean Vuillemin. Hardware speedups in long integer multiplication. *Computer Architecture News*, 19(1):106–114. 1991.

[20] Mark Shand and Jean Vuillemin. A hardware implementation for fast RSA Cryptography. *To appear*, 1993.

[21] Hervé Touati. Perle1DC: a C++ Library for the Simulation and Generation of DECPeRLe-1 Designs. *PRL Technical Note 4*, Digital Equipment Corp., Paris Research Lab., 85 av. Victor Hugo, 92563 Rueil Malmaison, France, 1992.

[22] Jean E. Vuillemin. Contribution à la résolution numérique des équations de Laplace et de la chaleur. *PRL Report 16*, Digital Equipment Corp., Paris Research Lab., 85 av. Victor Hugo, 92563 Rueil Malmaison, France, 1992.

[23] Xilinx, Inc. The Programmable Gate Array Data Book. *Product Briefs, Xilinx, Inc.*, 1987.

[24] J. Ziv and A. Lempel. A Universal algorithm for sequential data compression. *IEEE Trans. on Information Theory*, IT-23(3):337-343, 1977.