CERN 93-07

CERN 93–07
Electronics and Computing
for Physics Division
20 December 1993

ORGANISATION EUROPÉENNE POUR LA RECHERCHE NUCLÉAIRE
CERN EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

THE MPPC PROJECT
FINAL REPORT

Massively Parallel Processing Collaboration

Editor: F. Rohrbach

GENEVA
1993

© Copyright CERN, Genève, 1993

Propriété littéraire et scientifique réservée pour tous les pays du monde. Ce document ne peut être reproduit ou traduit en tout ou en partie sans l'autorisation écrite du Directeur général du CERN, titulaire du droit d'auteur. Dans les cas appropriés, et s'il s'agit d'utiliser le document à des fins non commerciales, cette autorisation sera volontiers accordée.

Le CERN ne revendique pas la propriété des inventions brevetables et dessins ou modèles susceptibles de dépôt qui pourraient être décrits dans le présent document; ceux-ci peuvent être librement utilisés par les instituts de recherche, les industriels et autres intéressés. Cependant, le CERN se réserve le droit de s'opposer à toute revendication qu'un usager pourrait faire de la propriété scientifique ou industrielle de toute invention et tout dessin ou modèle décrits dans le présent document.

Literary and scientific copyrights reserved in all countries of the world. This report, or any part of it, may not be reprinted or translated without written permission of the copyright holder, the Director-General of CERN. However, permission will be freely granted for appropriate non-commercial use.

If any patentable invention or registrable design is described in the report, CERN makes no claim to property rights in it but offers it for the free use of research institutions, manufacturers and others. CERN, however, may oppose any attempt by a user to claim any proprietary or patent rights in such inventions or designs as may be described in the present document.

ISBN 92-9083-056-5

CERN 93–07
Electronics and Computing
for Physics Division
20 December 1993

ORGANISATION EUROPÉENNE POUR LA RECHERCHE NUCLÉAIRE
CERN EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

THE MPPC PROJECT
FINAL REPORT

Massively Parallel Processing Collaboration

Editor: F. Rohrbach

GENEVA
1993

CERN–Service d'information scientifique–RD/918–2200–décembre 1993

ABSTRACT

We report on the work done in massively parallel processing with a view to studying possible solutions for extracting interesting high-energy physics particle events at future high-luminosity hadronic colliders operating in the TeV energy domain. We concentrate on a special Single Instruction Multiple Data (SIMD) architecture: the Associative String Processor (ASP). The Massively Parallel Processing Collaboration (MPPC) Project, grouping nine European institutes, was launched by CERN to carry out this R&D programme. This report, written by partners of the MPPC collaboration, describes the main results achieved at the end of the project: construction of ASP machines, parallel software development and application studies in high-energy physics and in other fields of science. A final, positive assessment of the ASP concept has been made by the Collaboration.

CONTRIBUTORS

S. Anvar[1], E. Augé[2], A. Basso[3], J.-L. Bertrand[2], R. Bishop[4], R. Bock[5], J. Bogdany[6], P. Borgeaud[1], J.C. Brisson[1], F. Bugeon[1], G. Burgun[1], D. Calvet[1], A. Ducorps[2], F. Dufaux[3], J. Feyt[5], O. Gachelin[1], M.-N. Gaujour[7], Ph. Heusse[2], T. Higgins[4], M. Izycki[8], I. Jalowiecki[9], A. Krikelis[4], M. Kunt[3], B. Lavigne[2], J. Lancaster[4], R.M. Lea[4,9], H. Le Provost[1], C. Maillet[1], M. Martin[8], M. Mur[1], G. Odor[6], L. Orsini[5], J.C. Raoul[1], V. Robert[1], F. Rohrbach[*5], A. Ster[6,8], B. Thooris[1] and G. Vesztergombi[6]

[1] CEA-CEN Saclay (France), [2] LAL, Université Paris-Sud, IN2P3-CNRS, Orsay (France), [3] EPFL - Ecole Polytechnique Fédérale de Lausanne, EPFL/LTS (Switzerland), [4] ASPEX-Microsystems Ltd., Uxbridge (United Kingdom), [5] CERN, Geneva (Switzerland), [6] KFKI, Budapest (Hungary), [7] Thomson-TMS, Saint-Egrève (France), [8] Université de Genève - Faculté des Sciences (Switzerland), [9] Brunel University, Uxbridge (United Kingdom).

* Spokesman

FOREWORD

The MPPC Project, originally set up in 1989, was approved in 1990 and a Memorandum of Understanding for the execution of the project was signed by:

Main Partners*

M. Aymar	Chef de la Direction des Sciences de la Matière (DSM), CEA-CEN Saclay, France
M. Davier	Directeur du Laboratoire de l'Accélérateur Linéaire d'Orsay, LAL, Université Paris Sud, IN2P3 CNRS, Orsay, France
W. Hoogland	Director of Research, CERN, Geneva, Switzerland
Prof. R.M. Lea	Chairman and Managing Director, ASPEX-Microsystems Ltd., Uxbridge, United Kingdom
P. Lehmann†	Directeur de l'IN2P3, IN2P3-CNRS, Paris, France

Associated Partners*

Mrs M.-N. Gaujour	CCD Marketing Manager, Thomson-TMS, France
Prof. M. Kunt	Laboratoire de Traitement des Signaux, Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland
Prof. R.M. Lea	Brunel University, Uxbridge, United Kingdom
Prof. M. Martin	Université de Genève - Faculté des Sciences, Switzerland
G. Vesztergombi	KFKI, Budapest, Hungary, a new associated partner from 1991

We underline the importance of collaborating with specialists outside the high-energy physics domain for this project. In particular, the importance of the collaboration with experts in signal and image processing like the LTS at EPFL must be emphasized.

A first status report was presented to the Detector Research and Development Committee (DRDC) at CERN at the beginning of 1991. The Project was terminated at the end of 1992. Although all the projected goals were not reached, the main objectives have been achieved: the construction, installation and running in each Main Partner institute of a fine-grain multiprocessor Associative String Processor (ASP) machine, and a broad study of applications of the ASP architecture for real-time processing in high-energy physics and in other fields of research. This work led to a preliminary, positive assessment of the ASP concept.

---

* The Main Partners collaborated in the full MPPC programme (hardware and software) as opposed to the Associated Partners, not committed to the development of the ASP hardware.

† Deceased.

Ad ASTRA per ardua...

CONTENTS

List of Contributors v
Foreword vii

1. INTRODUCTION 1

2. THE ASSOCIATIVE STRING PROCESSOR (ASP) CONCEPT 3
 2.1 The basic ASP chip 4
  2.1.1 The associative processing element (APE) 4
  2.1.2 Chip implementations 5
 2.2 The hybrid module 6

3. MACHINE ARCHITECTURE 9
 3.1 The ASP machine architecture 10
 3.2 The ASP global bus 18
 3.3 The low-level ASP controller (LAC) 18
 3.4 The ASP boards 20
  3.4.1 The ASPA card (chips) 20
  3.4.2 The HASPA card (modules) 22

4. THE ASP EMBEDDED NODE (ASPEN) 24

5. THE ASTRA MACHINE SOFTWARE 26
 5.1 Writing an ASTRA application 27
 5.2 Software development tools: the compilers 28
 5.3 Operating system tools and run-time libraries 28
 5.4 ASP documentation 29
  5.4.1 ASPEX documentation 29
  5.4.2 CERN documentation 29

5.5 Graphics tools 30
5.6 Other ASTRA programming methods 30
5.7 An example of applications programming on the ASTRA machines........ 31

6. A CCD INTERFACE TO ASTRA 31
6.1 Description of the CCD readout and its interface to the ASTRA machine 31
6.2 Test results of the CCD-ASTRA system 33
6.3 A CCD interface for the MPPC machine 33

7. APPLICATIONS 35
7.1 Application studies on a VASP simulator 35
7.1.1 Tracking and calorimetry for the SDC level-2 trigger 35
7.1.2 The LHC high-transverse-momentum muon second-level trigger 37
7.1.3 The transition radiation tracking (TRT) detector for LHC. Simulations for applications of ASP modules for a 100 kHz trigger 40
7.1.3.1 Feature extraction 40
7.1.3.2 Benchmark definition 41
7.1.3.3 Benchmark results 42
7.1.3.4 Discussion of the ASP implementation 43
7.1.4 Image coding applications 45
7.1.4.1 Image compression based on a Gabor-like wavelet transform 45
7.1.4.2 Neural autoassociation for image compression: a massively parallel implementation 48
7.1.4.3 Conclusions concerning the use of ASP for image compression 51
7.2 Application studies on ASTRA machines 52
7.2.1 Online data-processing in a high-energy physics experiment 52
7.2.2 The SDC second-level trigger 53
7.2.3 Image processing for peak-finding from cluster data 54
7.2.3.1 The peak-finding algorithm 54
7.2.3.2 Algorithm timing results 56
7.3 ASPEN evaluation in the NA48 experiment 56

8. PERSPECTIVES 57
8.1 A new chip: the VASP-128 57
8.2 Development of a two-dimensional ASP 58

9. SUMMARY AND CONCLUSIONS 60

APPENDIX 63

A.1 FIRST EXAMPLE: WRITING THE SKELETON OF AN ASTRA PROGRAM 63

A.2 SECOND EXAMPLE: AN EXTENDED PROGRAM, ARRAY OF SUMS 67
A.2.1 Passing data between HAC and IAC 68
A.2.2 Passing data between IAC and LAC 68
A.2.3 The pSums implementation 69
A.2.4 The pSums makefile 75

References 77

1. INTRODUCTION

The starting point of the Massively Parallel Processing Collaboration (MPPC) Project [1] is to be found back in 1988: at that time, it was already recognized that there would be a strong need for real time sophisticated triggering systems for the detectors used with the hadron colliders of the future. Interesting, rare physics events for new discoveries will have to be extracted in real-time from the enormous physical background due to the high value of the total proton–proton cross-section. The task of the triggering system would consist of extracting essential features for tagging interesting physics events from the massive amount of raw information (many megabytes) delivered at a high rate (40–66 MHz) by the multimillion detector channels surrounding a proton–proton collision area.

This trigger task was foreseen as a three-step process called level-1, level-2, and level-3 trigger (see Fig. 1).

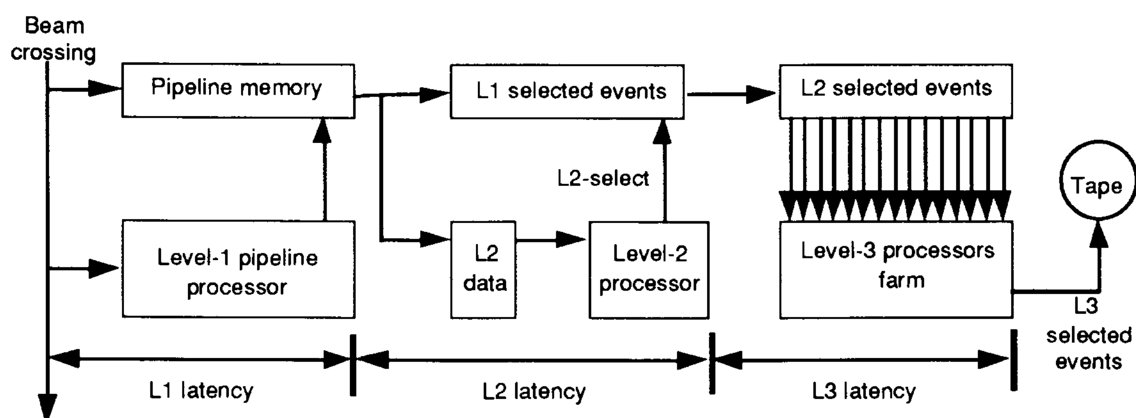


Figure 1 Three-level trigger organization

Each level is characterized by an acceptable rate and by a latency time. The level-1, mainly hard-wired processor is able to sustain the rate of one event each 16–25 ns with a latency of a few microseconds. Based on a local data analysis, the level-1 process is supposed to reduce the rate by a factor of 1000–5000, allowing an average rate to the level-2 of one event every 10–50 μs. Events selected by the level-1 process will be stored in a buffer memory; the accepted latency, which in turn determines the size of this buffer, will be between 50 and 200 μs. The level-2 process will be based on a rough, global, topological analysis of the event and energy threshold combinations. As far as this is physics-dependent, the level-2 algorithms need to be flexible, i.e. programmable. The rejection factor is estimated to be between 50 and 100. The level-3 process will be handled by a farm of 1000–5000 high-speed microprocessors on standard boards which

make a detailed physical analysis of each selected level-2 event. The selected level-3 events are then recorded on tape at the rate of about 10 events per second.

In 1988, no tools were available or in sight for performing the complex task of the level-2 trigger.

In 1989, a new Single-Instruction Multiple Data (SIMD) architecture—the Associative String Processor (ASP) devised by ASPEX Microsystems Ltd.—was identified as potentially able to solve the bottleneck foreseen for a level-2 trigger. This led to a collaboration which was established to study potential applications of ASP in High-Energy Physics (HEP) second-level-trigger applications. With this task in mind, the collaboration set up the MPPC Project [2].

This project was focused on two topics: the evaluation of algorithms using massively parallel processors for solving difficult triggering problems, and the learning of the required know-how in designing, building, and using a machine equipped with a large number of parallel processors.

As a first goal, the collaboration dedicated a large fraction of its manpower to establish algorithms and to execute them on the ASP simulator running on a Sun workstation. Later, algorithms were ported on the real machine.

The second pole of the activity concentrated on the specification, design, construction, and commissioning of a machine based on the ASP concept: thousands of integrated processing elements arranged in a string and providing a flexible and scalable architecture endowed with an intelligent and powerful communications network.

A first status report on the MPPC Project was presented to the Detector Research and Development Committee (DRDC) at CERN in 1991 [3]. The main project results have been regularly reported in international conferences and workshops [4–7]. The present report gives an account of all the work done in hardware and software and the main application results obtained by the collaboration during the project.

The first implementation of the machine, ASP System Test-bed for Research and Application (ASTRA)* , is now operational at CERN, Orsay, Saclay, and ASPEX. As planned, they are used as test benches for online application algorithms and provide real timing values. An ASTRA user-friendly environment has been created at CERN and is available for physicists interested in testing fast, massively parallel, triggering algorithms on the ASTRA machine.

The R&D MPPC Project at CERN ended in 1992 with the installation of the first ASTRA machine. The development of ASP machines continues at CEA-CEN Saclay and at ASPEX Microsystems Ltd.

To be closer to the real use of ASP in HEP experiments, ASPEN, a prototype of an embedded machine has been designed at Saclay. This machine, based on a real-time, distributed processing power philosophy, associates conventional microprocessors (DSP) with a string of ASPs to obtain maximum flexibility and efficiency.

---

* The ASTRA machine was formerly known as the ASPA machine.

The final goal of these studies is to run online applications in physics experiments.

2. THE ASSOCIATIVE STRING PROCESSOR (ASP) CONCEPT

The ASP consists of a string of Associative Processing Elements (APEs). ASP devices belong to the Single-Instruction, Multiple-Data (SIMD) class of parallel processors. One instruction is simultaneously applied to a large number of identical Processing Elements (PEs), each storing a different datum. Each element possesses its own local memory and is able to perform elementary instructions. To provide the ability for data-dependent processing, each PE is able to make an association between its own stored data and a key pattern which is presented in parallel to all PEs. This associative parallel strategy forms the basis of the ASP architecture concept: the conventional successive addressing of each PE is replaced by an associative global data access (content addressing). All the elements share the same bus, and are also interconnected with high efficiency in order to transfer flags and data.

ASP modules are the basic blocks for the construction of programmable, scalable, fine-grain SIMD machines.

Synchronous and asynchronous communication between APEs is provided through a dynamically reconfigurable inter-APE communications network, with a string topology. Parallel processing is performed on active subsets of APEs, preselected in order to run the steps of dedicated algorithms. The architecture is reconfigurable by programming. The string can be arranged in a loop through the controller and segmented with bypass possibilities. It is fault-tolerant: blocks of faulty APEs may be deactivated without breaking the string.

Loading and unloading data in the string is done on a 32-bit word or byte exchange basis. A typical set of machine operations, which can be considered as its basic cycle, consists of a sequence of suboperations:

- a search then a tag of matching APEs, with a possible shift of the tag along the string, accompanied by a clear of bits, bytes or words;
- the activation of tagged APEs or of a pattern of APEs related to them that will participate in the next step;
- the execution of Read or Write suboperations in the activated elements.

The architecture is scalable to hundreds of thousands of APEs, due to high integration (VLSI/WSI) and low power consumption (≈1 mW per APE). The target cost is low ($1 per APE) leading to the possibility of massive integration.

2.1 The basic ASP chip

The chip used in this project is the VASP-64 (see Fig. 2). The VASP-64 chip contains a string of 64 APEs connected to a programmable intercommunications network allowing the APEs to communicate together. The network is used to connect APEs together in a string. A logic block handles the commands (control bus) and the data (data and activity bus) required to execute an operation, and allows one to select a bit/byte or word data format and to mask the data or activity bits.

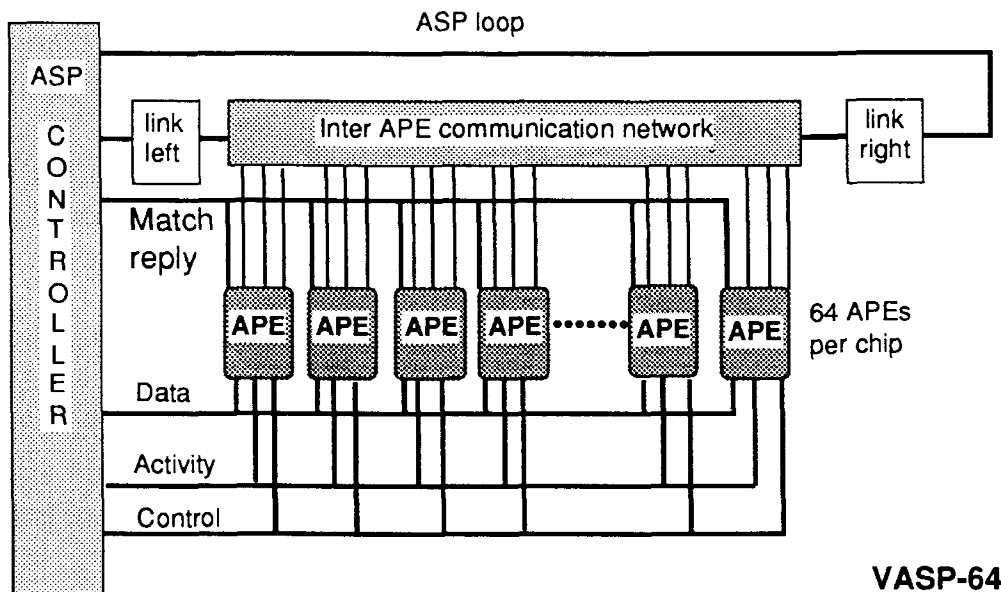


Figure 2 The associative string processor

VASP chips may be cascaded to build a multichip APE string of any length required by the application. Programmable block links are interspersed in the communications network at eight APE intervals, allowing the string to be partitioned into segments between which communication may be isolated/linked to the neighbouring segment.

2.1.1 The associative processing element (APE)

The internal structure of an APE is shown in Fig. 3. It contains two main registers, a 64-bit-wide data register and a 6-bit-wide activity register. The data register can be written from the data bus and its contents can be read out on the same bus. The activity register may only be written. The contents of these registers can be compared with the state of the data and activity busses in a bit-parallel fashion. The result of the comparison, match or mismatch, is stored in one or both of the tag registers 'Tr1' or 'Tr2'.

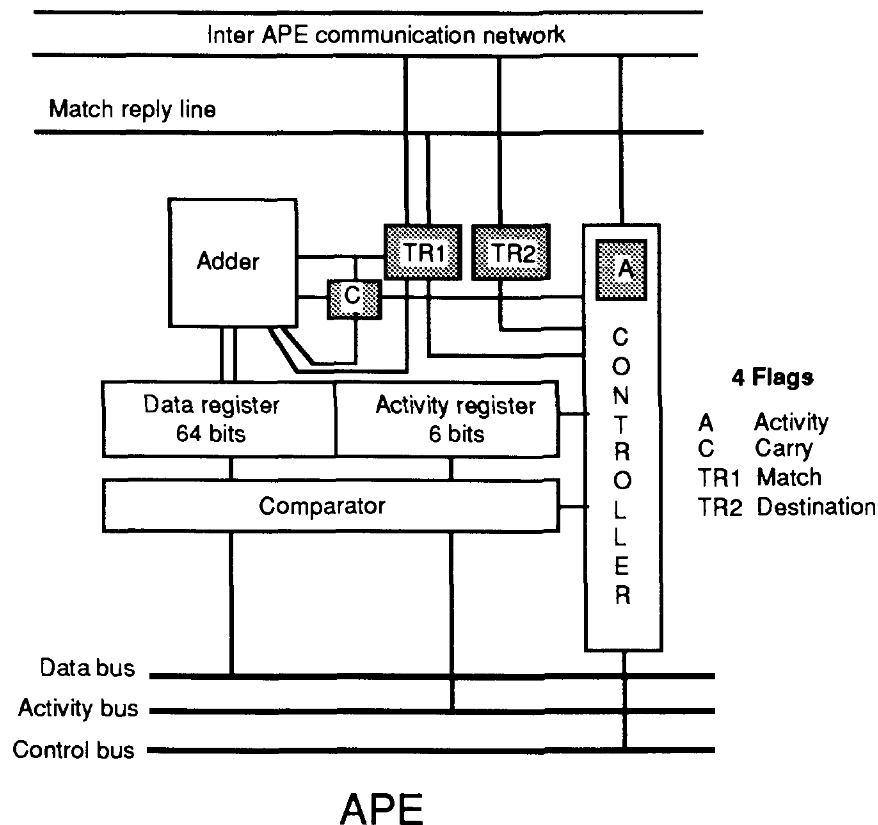


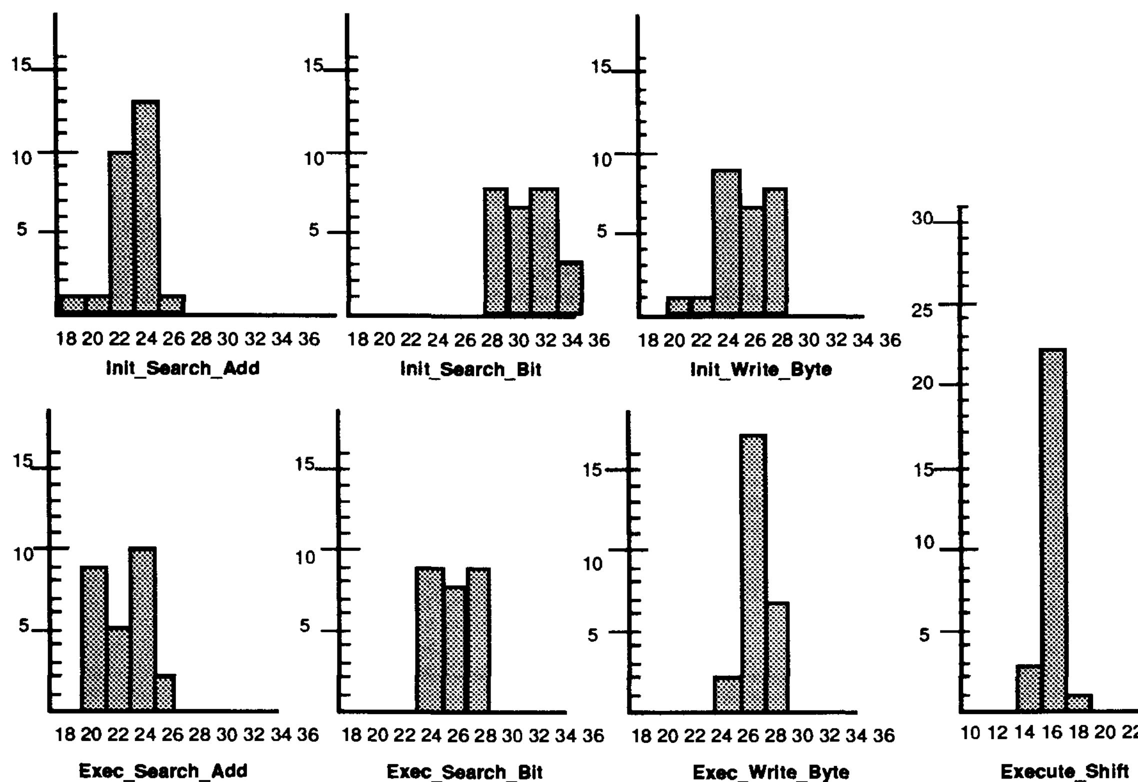
Figure 3 The associative processing element

The serial full adder is able to add two one-bit values representing the contents of two bits of the data register, selected by a search operation, together with the contents of the carry register. The sum is stored in 'Tr1' and the new carry overwrites the contents of the carry register. Other add operations are available between the data bus, the data register, or the flag 'Tr1'.

2.1.2 Chip implementations

The first implementation of the VASP-64 was done by ES2† in a CMOS 2 µm dynamic technology using an electron-beam implantation with a time slot of 40 ns in August 1990. The mass production using a mask implantation showed a degradation of the characteristics. The measured time slot, 70 ns, was considered too slow and a new design was started with Hughes [8] on a Silicon On Sapphire (SOS) 1 µm static technology (see results for VASP-64/H1-B2 Fig. 4). The present implementation (H1-B3) on a 0.8 µm design rule gives a time slot of 25 ns for the slowest subinstruction and has been available since spring 1993.

† ES2, European Silicon Structures, Zone industrielle, 13106 Rousset CEDEX, France.



Horizontal scale: time slot in nanoseconds
Vertical scale: number of chips

Figure 4 The ASP status: H1-B2 chip speed results

2.2 The hybrid module

A hybrid module (HASP) was designed to allow for maximum processor element density and maximum direct parallel interfacing via conventional electronics to the readout of particle detectors. For this task, dense packages of ASPs had to be constructed. They were based on a modular design using hybridization on insulators of the VASP-64 chips. Three designs were successively developed.

In September 1990 a preliminary design for a HASP with 1024 APEs and four I/O channels was completed (see Fig. 5). Further studies showed that, owing to the large number of pins, an expensive custom package would be required and space for implementing four input channels per HASP was not available on the circuit board.

The revised design had two I/O ports which could be configured as 2×512 APE substrings or one substring and allows bypass of 64 or 512 APE blocks (see Fig. 6). This revised design was targeted to a standard 184-pin package. In addition to the VASP chips the HASP contained glue buffers and Programmable Array Logic (PAL) devices.

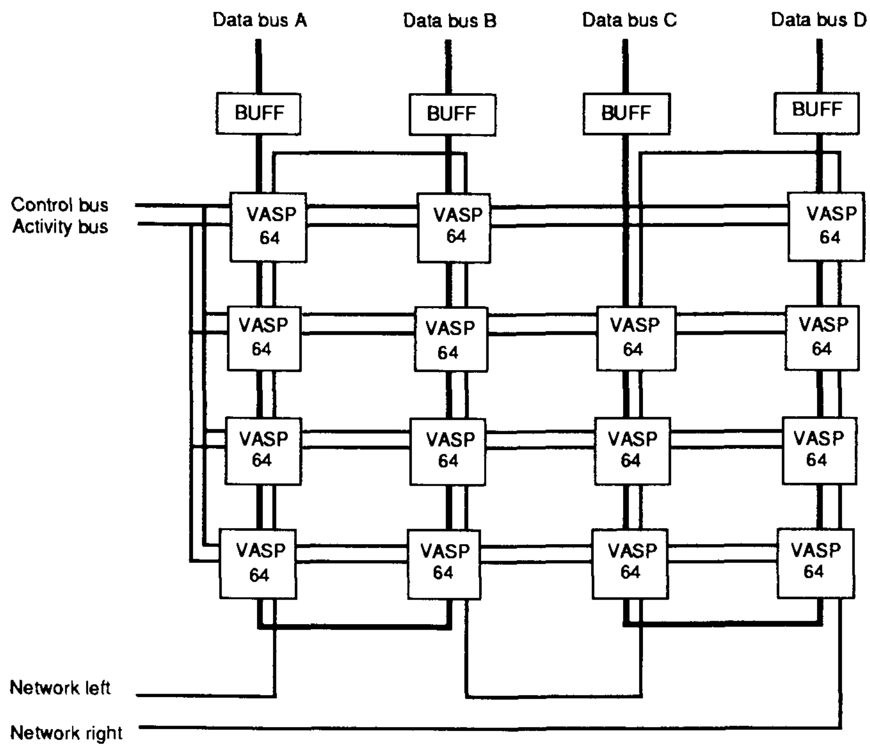


Figure 5 The hybrid ASP module: first organization

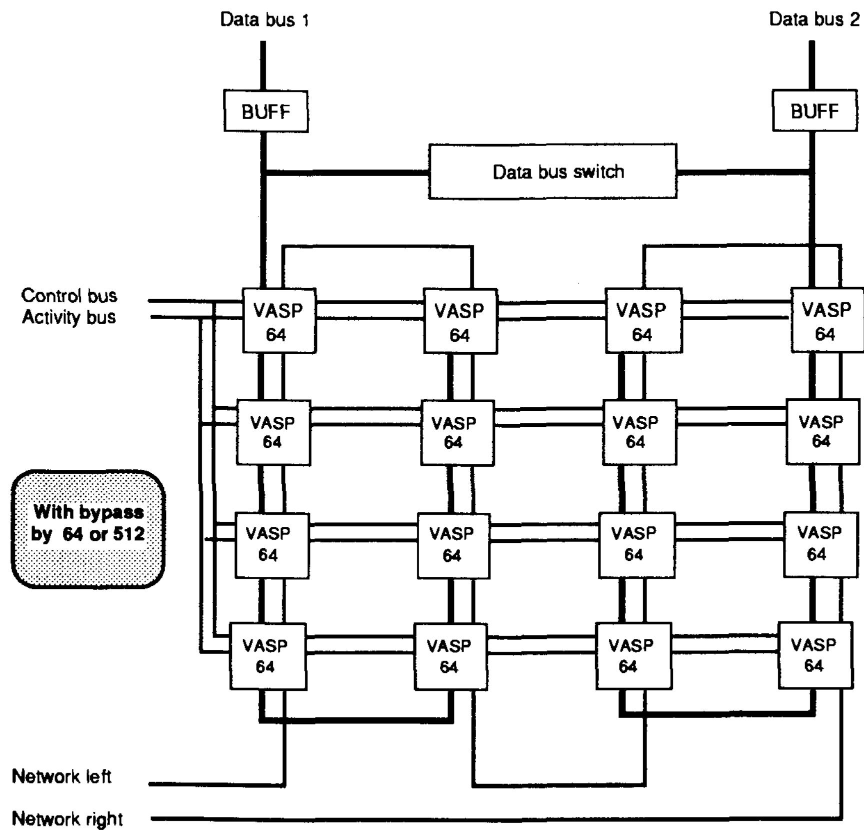


Figure 6 The hybrid ASP module: second organization

A thermal analysis of the HASP design had been undertaken. The results showed that a reasonable airflow (1 m per sec) across the surface of the HASP would hold the device junction temperature below 70 °C.

The design was issued to PolyCon, an American hybrid manufacturer, but in December 1991, for internal reasons, PolyCon resigned not far from the target. Then Hughes, which was in charge of the SOS version of the VASP-64, accepted to redesign and to produce the hybrid. Taking advantage of this unfortunate event, this third design used the results of an ASP prototype machine built in the mean time. This new HASP contained 16 VASP-64 chips (1024 APEs) together with two data ports. Each 32-bit data port had bidirectional buffers; the control bus and activity bus were internally buffered. Bypass and selection logic were included in the hybrid (see Fig. 7).

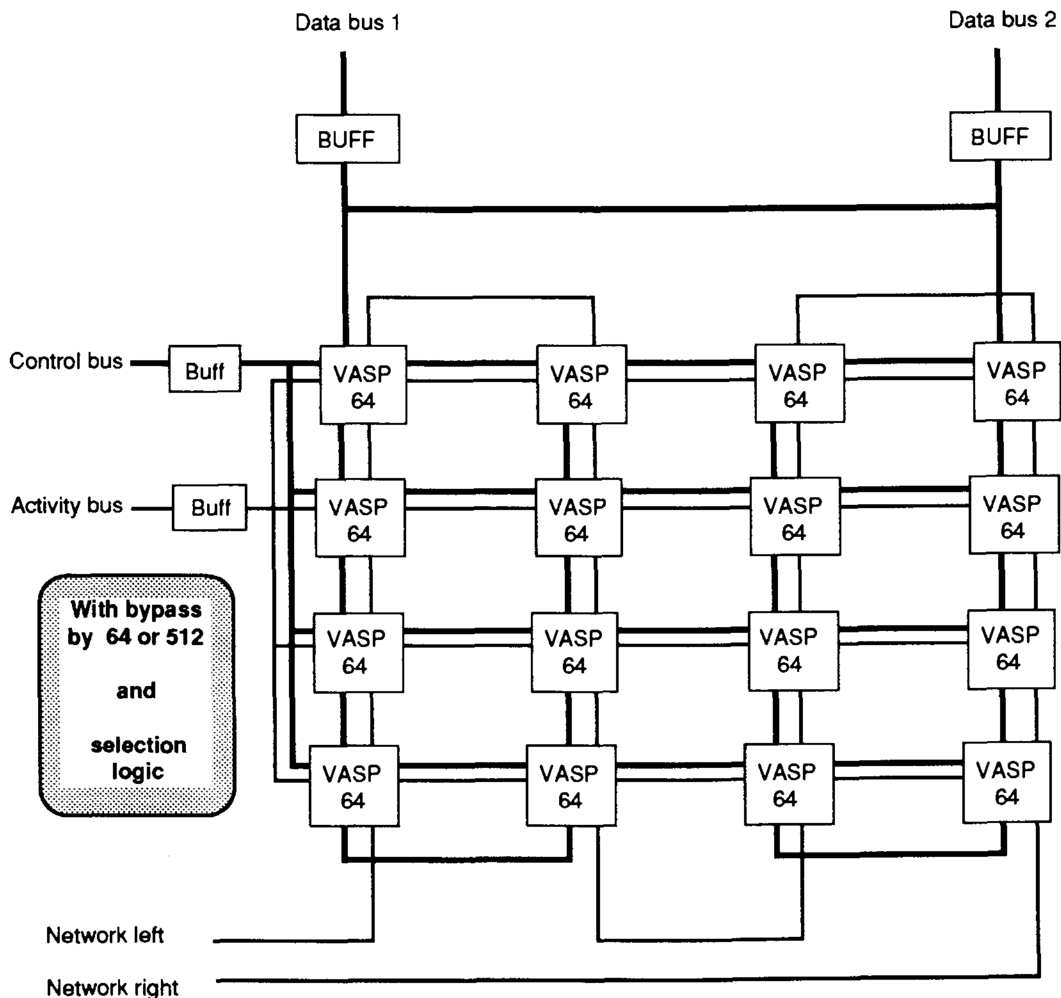


Figure 7 The hybrid ASP module: final organization

Unfortunately, Hughes also failed to manufacture this hybrid. Therefore the collaboration stopped the hybrid studies and concentrated on the discrete chip machine.

3. MACHINE ARCHITECTURE

Figure 8 shows the well-known Single Instruction applied to Single Data (SISD) general architecture for a microprogrammed processor. To increase the processing power it is possible to design parallel architectures by using many processing devices (Arithmetic and Logic Unit or ALU) working in parallel on an array of data and leading to a Single Instruction applied to Multiple Data (SIMD) structure (see Fig. 9). In a multi-ALU machine two types of data can be considered, scalar and vector. Each ALU works simultaneously on its own vector data, whereas scalar data is broadcast over all ALUs.

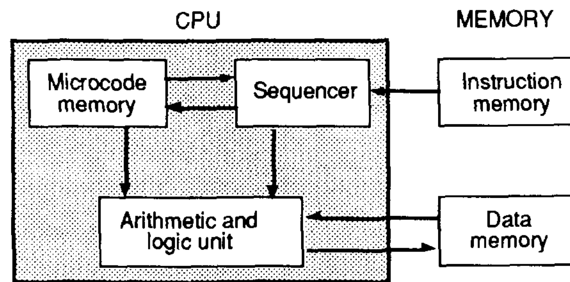


Figure 8 Structure of a microprogrammed processor (SISD)

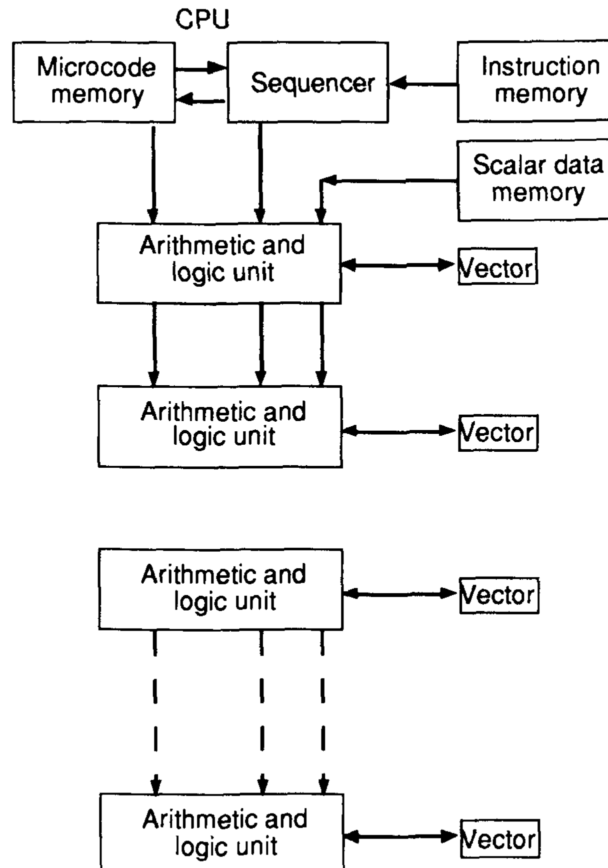


Figure 9 Structure of a vectorized processor (SIMD)

Some limitations slow down the speed of such a machine. The first bottleneck is the competition to get simultaneously vector data in the main memory; the second problem is to disable parts of the ALU depending on previous results during the processing.

The ASP chip is ideally suited to build SIMD machines in which the ASP string is used as a multi-ALU. The ASP structure avoids some difficulties of a SIMD machine: vector data are distributed and kept in each ALU (APE), access to each APE is an associative process, one can work with a subset of the string driven by the data.

3.1 The ASP machine architecture

In order to evaluate on real machines the ASP concept for online, real-time applications, the MPPC collaboration worked to design and construct four multipurpose, real-time ASP machines (one for each main partner). Two types of ASP machines were designed, one with the ASPA board (2 K‡ processors) using individual chips (known as the ASTRA machine), and the other one with the HASPA board (8 K processors) using hybrid modules.

Each machine is built in a pseudo-VME crate using a triple-height eurocard board (see Fig. 10). They are hosted by a UNIX workstation linked to a VME crate through a SVIC 7213 and a VIC 8250 [9], [10]. The VME crate is equipped with a standard VME processor (FIC 8232 [11]), a single-board, low-level controller (LAC) optimized to generate all possible ASP instructions, and one or more VME processing boards containing an array of ASP chips (ASPArray or HASPArray boards).

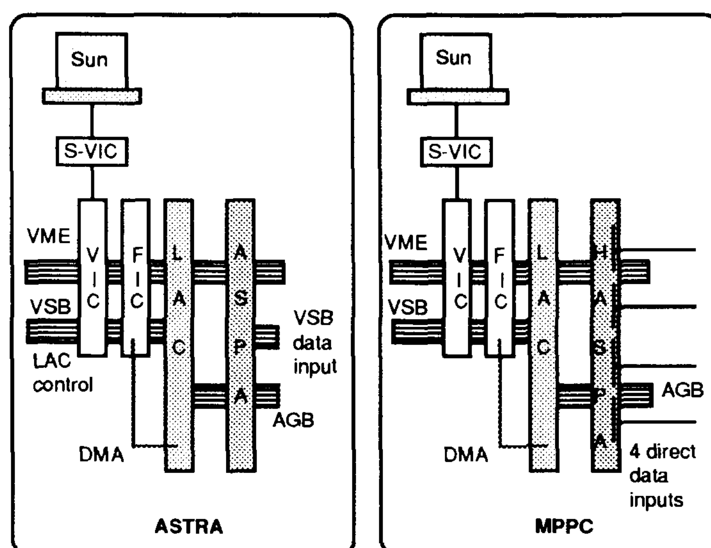


Figure 10 The ASP machine organization, ASTRA machine left, MPPC machine right.

‡ K is equivalent to 1024.

Figure 11 shows the architecture of the ASTRA machine and Picture 1 shows the machine itself working at CERN.

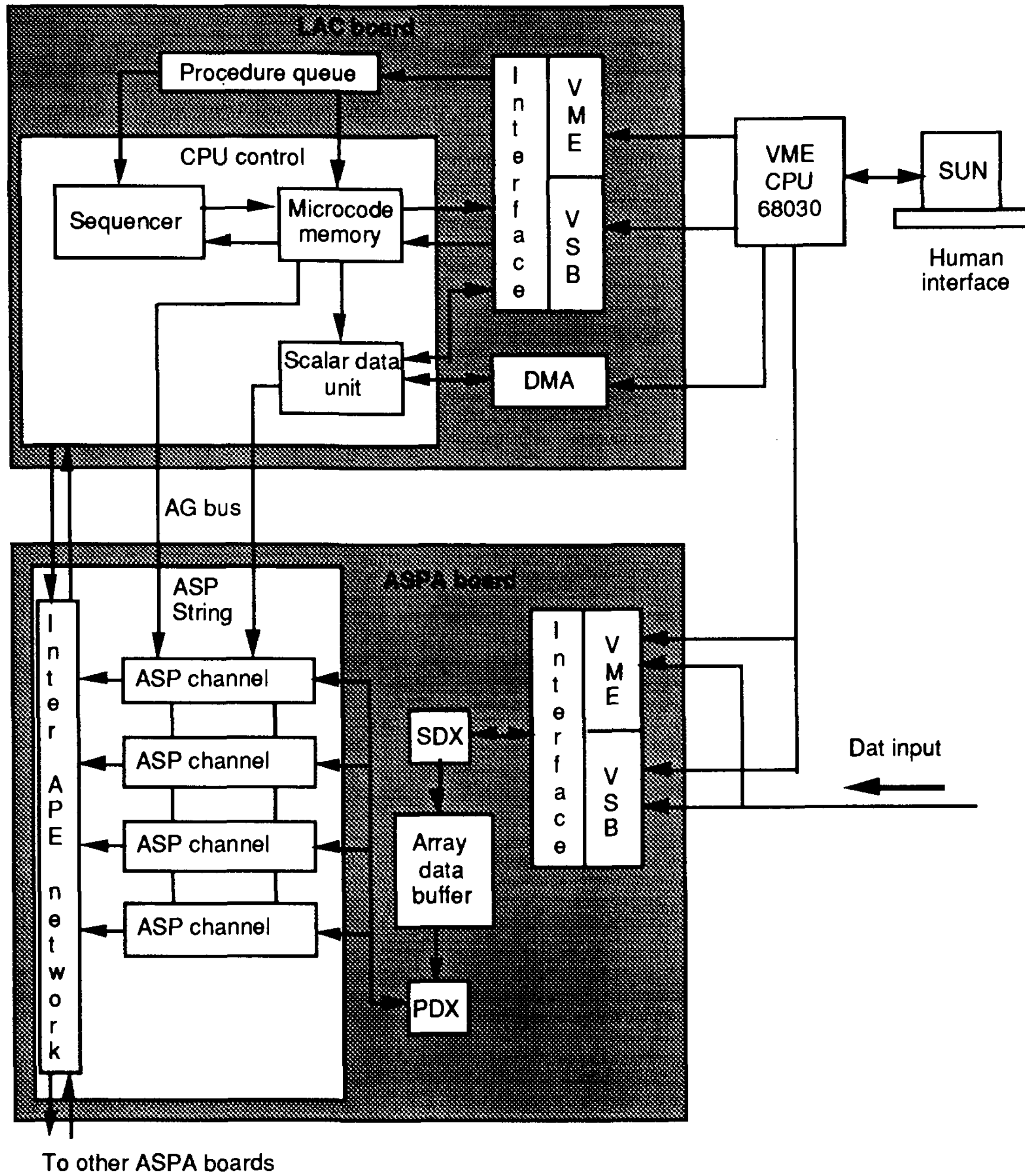
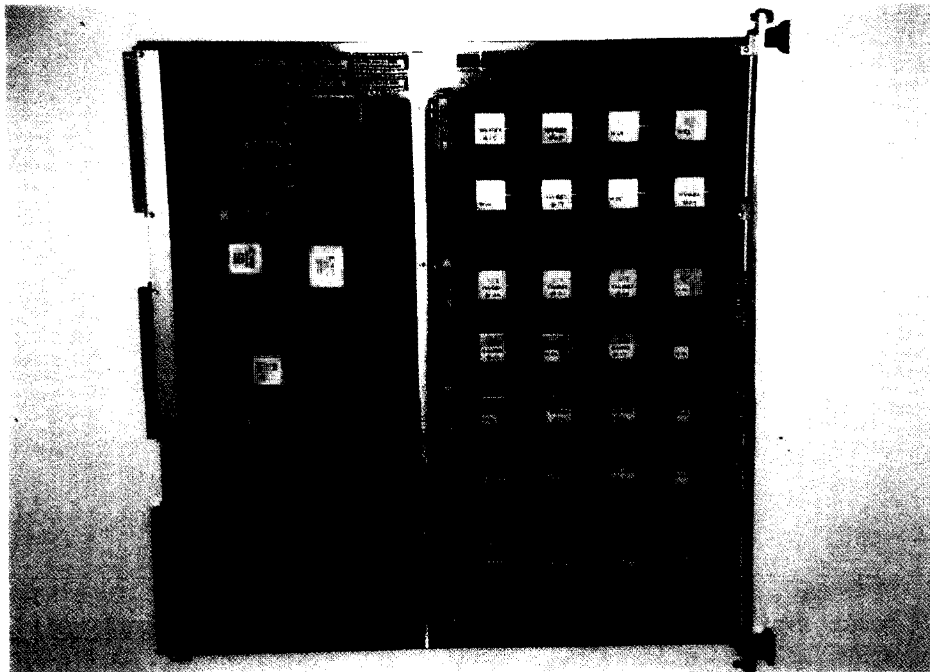


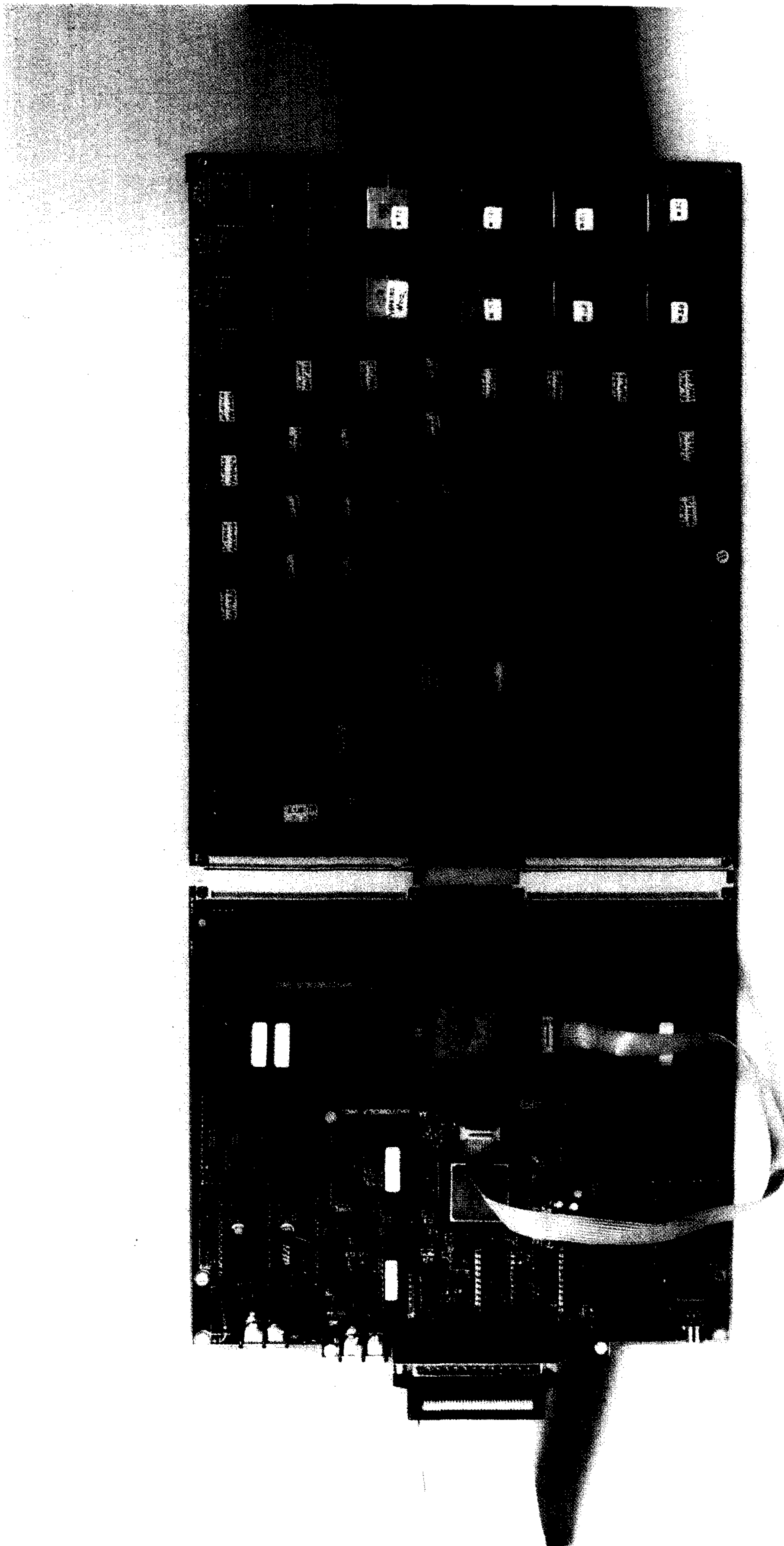
Figure 11 The ASTRA machine architecture

Picture 1. L. Orsini working on the ASTRA (ASPA) machine at CERN. The two extended VME boards, the LAC and the ASPA are clearly identified. Eight ASPA boards can be installed in the VME crate, making a 16K machine.

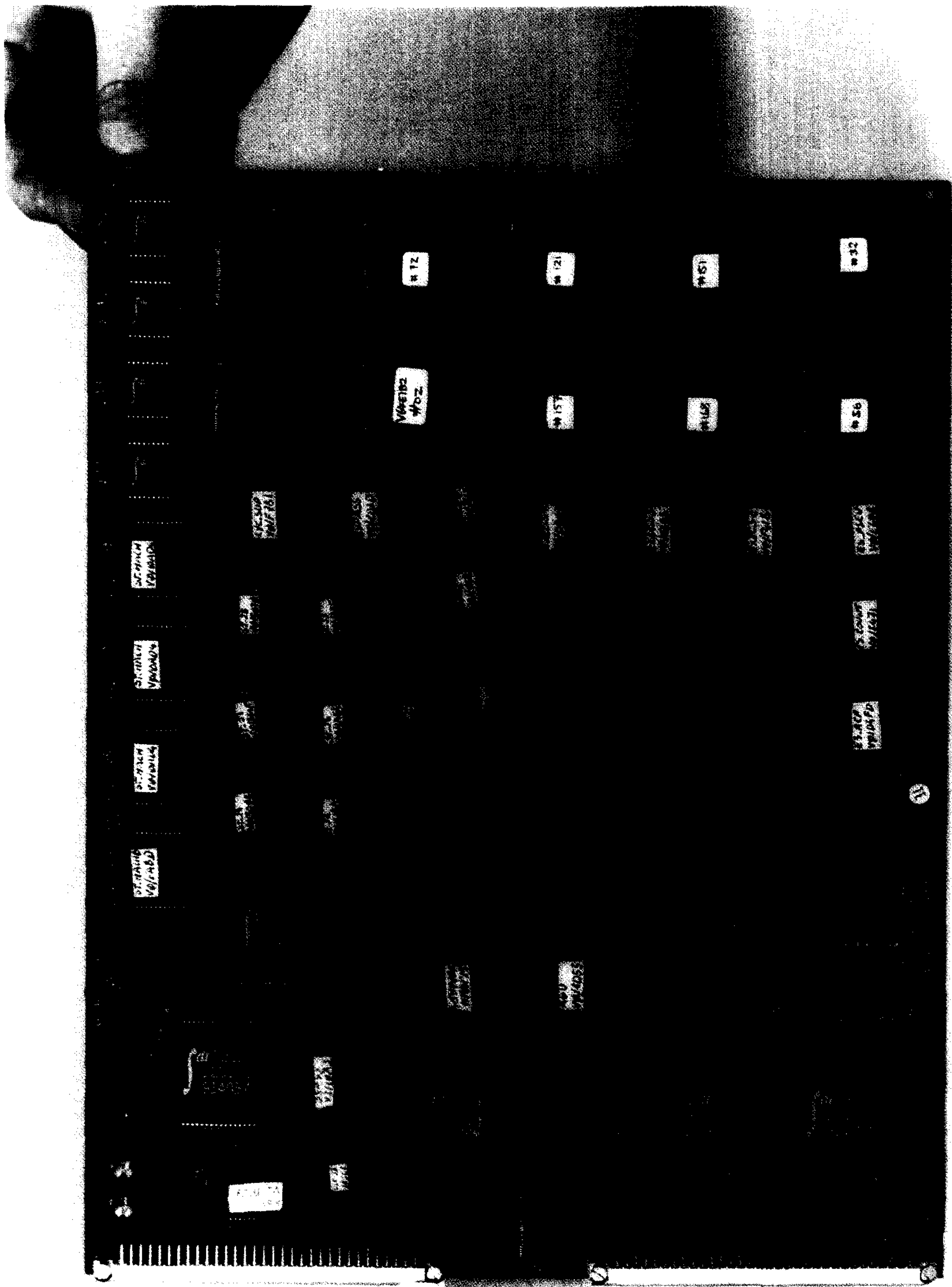
Picture 2. The LAC board



Picture 3. The ASPA board; the 32 VASP-64 chips are easily identified on the right.



Picture 4. The complete ASPEN prototype showing the ASP board (top) and the DSP board (bottom)



Picture 5. A close-up view of the ASPEN interface circuitry; the eight VASP64 ASP chips are clearly identified on the top.

The architecture of the MPPC machine is shown in Fig. 12.

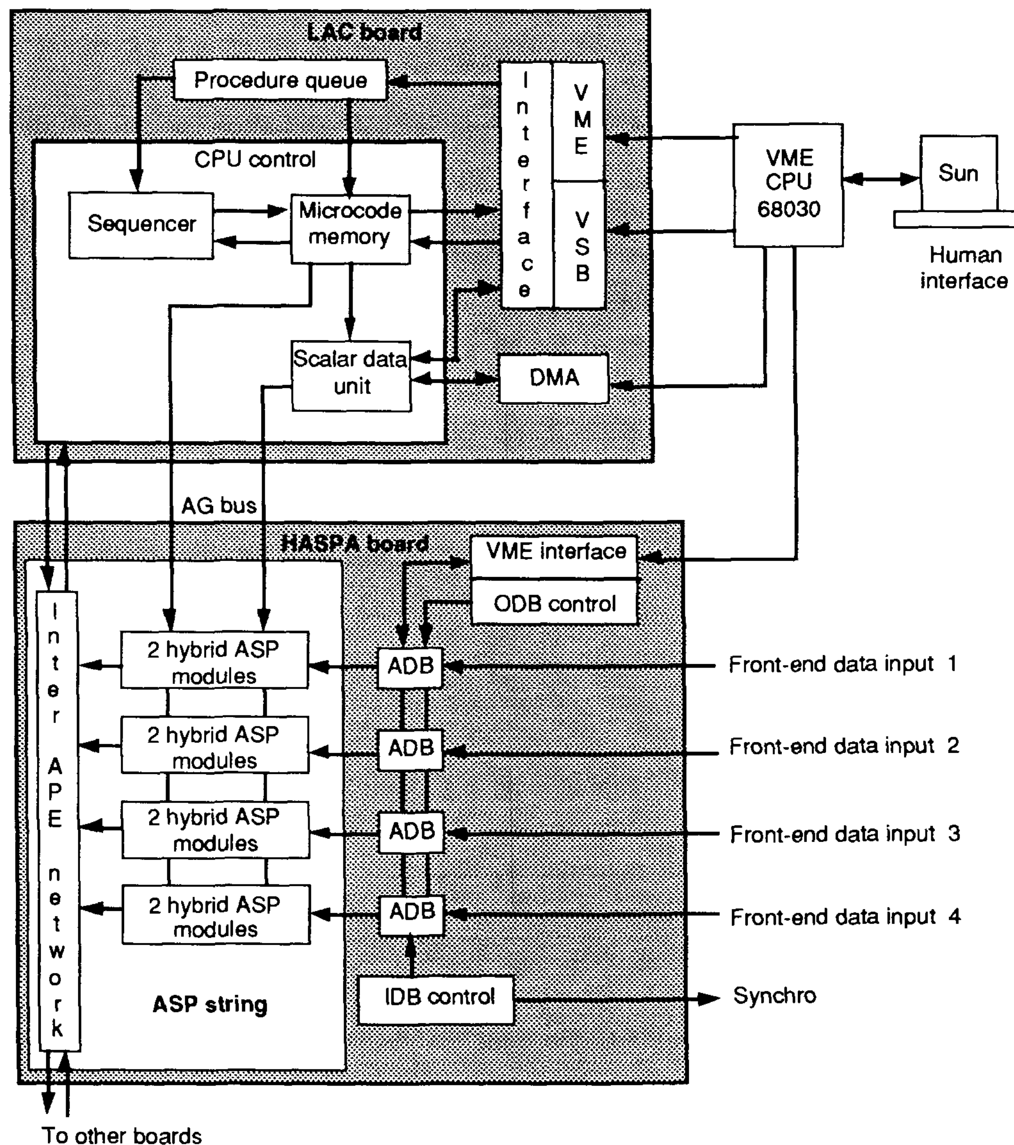


Figure 12 The MPPC machine architecture

For both machines, the Sun workstation (used as a human interface to the machine) is called a High-Level ASP Controller (HAC) and drives the FIC (a commercial 68030 VME CPU board) called Intermediate-Level ASP Controller (IAC). This FIC drives the ASP machine by three different accesses: VME and VSB busses, and a proprietary fast DMA channel. The ASP machine itself is composed of one Low-Level ASP Control board (LAC) and a maximum of eight ASP Array boards (ASPA or HASPA). The ASP machine must be considered as a vectored coprocessor of the 68030. On the LAC board, the microcode memory stores the algorithm procedures to be executed by the ASP string.

A bidirectional inter-APE network interconnects the LAC and the ASP boards in a double loop. Its purpose is direct communication between APEs; this network works in two modes: shift the content of the APE string over to the next string for comparison, or execute a remote activation of a selected APE.

The ASPA board contains 32 VASP chips (2048 APEs) allowing a machine up to 16K processors maximum. A parallel access to each board can be achieved by a direct connection to the VSB connector on the backplane through a small adaptor card. In this case each connected instrument must act as a master to generate the VSB protocol.

The HASPA board was designed to use the hybrid module described previously (see Fig. 7) which contains 16 dies of VASP-64 circuit. Each HASPA board contains eight HASP modules giving a total of 8192 APEs per board. Each pair of modules has its own Array Data Buffer (ADB) to allow a faster feed for data; four connectors allow the connection of external data acquisition logic. These connectors allow a data input rate of 320 Mbytes per second per board. The ADB memory is a double-port memory large enough to store more than one event so that it can be fed with the next event during the processing of the previous one.

3.2 The ASP global bus

The backbone of the ASP machine is a proprietary bus, the ASP Global bus (AGbus). The communication between the LAC and the ASP array is done by this AGbus on the P3 connector of the extended VME card.

The scalar data are transmitted on 32 lines in the form of a 32-bit binary word or in ternary mode[§] in case of bytes or bits. The activity bits use 12 lines and are always transmitted in ternary mode.

The instructions from the LAC are transferred over the AGbus in a compressed form (20 bits). They are expanded inside the ASP boards. The AGbus also carries synchronization signals and status (e.g. match). Four daisy chains are used between boards in the AGbus backplane to implement the inter-APE network.

3.3 The low-level ASP controller (LAC)

The Low-Level ASP Controller (LAC) card provides the environment to execute the ASP application program on one or more associated ASP boards using the AGbus. The LAC is controlled by higher level ASP controllers over its VME and VSB interfaces. A DMA Peripheral Bus (DPB) connects the IAC CPU card to the LAC directly.

[§] Allows the transfer of true, false, and don't care information.

The LAC block-diagram is shown in Fig. 13. The major parts are a CPU formed by a sequencer and a microcode memory to store the LAC operating system and the low-level procedures used in application programs. This Micro Instruction Buffer (MIB) is a 152-bit-wide memory, 64K words deep at the maximum, allowing a subinstruction rate above 40 MHz.

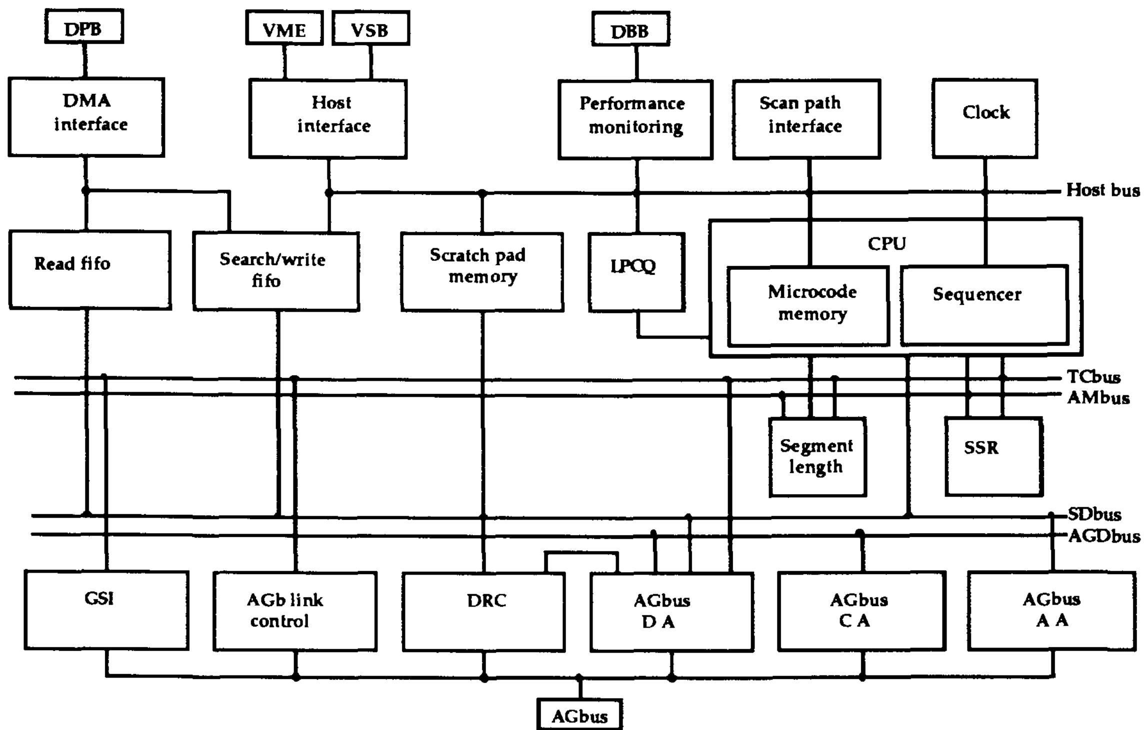


Figure 13 The LAC architecture

The Low-Level Procedure Control Queue (LPCQ) receives commands from the IAC to be executed by the ASP machine. Scalar data can be shifted left or right by the Scalar Shift Register (SSR), and can be stored in a high-speed scratch pad memory and in two fifos, the Search/Write fifo and the Read fifo. The Data Conversion Register (DCR) provides conversion to fit the ASP data representation (bit, byte or word). The AGbus Data Assembler (AGbus DA) assembles data words to be broadcast over the AGbus to ASP boards. The AGbus Control Assembler (AGbus CA) and AGbus Activity Assembler (AGbus AA) assemble a control word and activity bits to be broadcast also to ASP cards. The Global Status Inductor (GSI) interfaces the AGbus to the global status indicator. The AGbus Link Controller provides control and monitoring of the inter-APE network.

Many of these features may be accessed by the host; the host bus connects the host interface to the major parts of the card. Other internal busses interconnect LAC elements. The Test Condition bus (TCbus) connects test condition sources to the branch logic of the sequencer; the Address Modifier bus (AMbus) connects address modifier sources to the

address generation logic of the sequencer. The Scalar Data bus (SDbus) is formed by two 32-bit-wide data busses, the read (SDbus.rd) and the search/write (SDbus.sw) busses. The LAC Data bus (LACDbus), the AG Data bus (AGDbus) and the CTRLbus carry micro-order fields from the micro-program store to functional blocks of the LAC and through them to the AGbus. The scan path bus connects all registers of the board not accessible from the host bus into a serial bus. The performance monitor and the scan path interface allow monitoring and debugging of the machine through the Debug Board Bus (DBB).

The LAC card is a multilayer board with six layers of signal and two layers for ground and Vcc planes. The microcode memory is made in two banks and, using 8K or 32K pin compatible memories, it is possible to obtain 8K, 16K, 32K or 64K size. Most of the glue logic is done with Xilinx PGAs. The realization of the LAC board is shown in Picture 2.

3.4 The ASP boards

The two ASP array boards designed contain either ASP chips (ASPA) or modules (HASPA) and their associated Array Data Buffers (ADBs).

3.4.1 The ASPA card (chips)

The ASPA card block diagram is shown in Fig. 14 and the realization of the board is shown in Picture 3. The main part is the ASP Array (ASPA) comprising four ASP channels with eight VASP-64 chips per channel to give a total of 2048 APEs. Two sets of look-up memories are used to expand the AGbus instructions into two 16-bit subinstructions sent to the ASP chips and into a 32-bit pattern distributed over the card control bus.

The ADB comprises two dual-ported memory planes, providing one output and one input data channel for the ASP array. Each memory plane is divided into two pages, A and B, which allow simultaneous internal and external access. The ASP communicates with the ADB memory through the Global Data bus (GDbus) formed by two 32-bit-wide busses, the Global Data Input bus (GDIbus) and the Global Data Output bus (GDObus). The Primary Data exchange Address Logic (PDX-AL) generates the ADB addresses on the PDXbus during sequential 32-bit transfer between the ASP array and the ADB. The LAC microprogram may access ADB planes of a selected page for a PDX transfer independent of an SDX operation.

The host through a VME or a VSB interface may access the ADB plane of the other page simultaneously for a Secondary Data exchange (SDX) either in random access or in sequential access. SDX Address Logic (SDX-AL) is responsible for generating ADB addresses during sequential transfer.

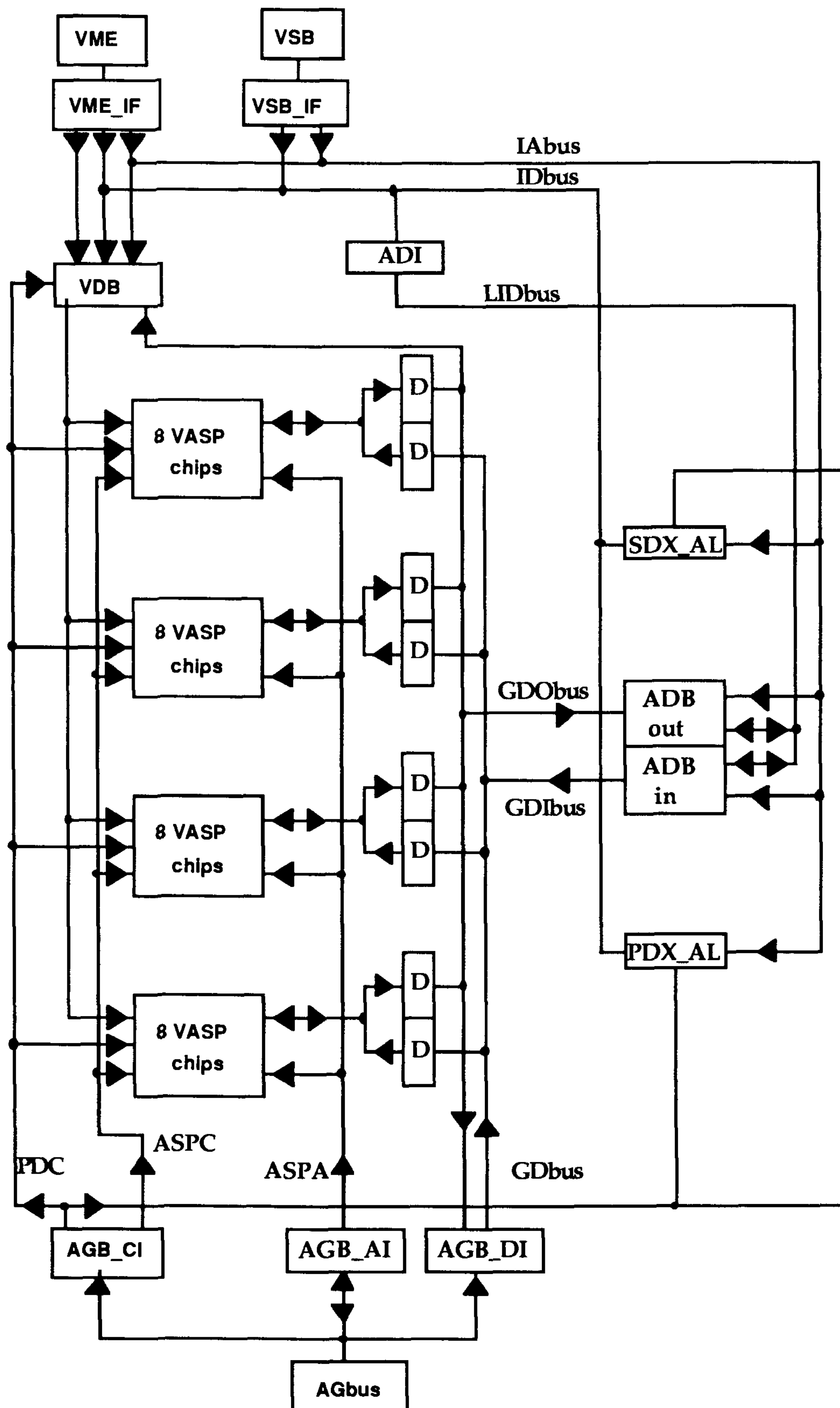


Figure 14 The ASPA card architecture

The IDbus is an internal data path for the VME or VSB interface, LIDbus is the internal data bus to input the ADB. Between these two busses, the ASPA Data Interface (ADI) is used to align data in a single byte, word (double byte) or long word (quad byte) for the host access to the ADB, and to perform conversion for the ASP ternary mode. The VASP Debugger (VDB) is a means for the host to monitor the internal state of a selected VASP-64 chip. The content of the selected chip is stored in a dual-port memory which can be subsequently read by the host.

The ASPA card is a multilayer board with six layers of signal and two layers for ground and Vcc planes. Most of the glue logic is done with Xilinx PGAs.

3.4.2 The HASPA card (modules)

The architecture of the HASPA board is roughly the same as that of the ASPA board but with a number of processors increased by a factor 4 and the implementation of four direct parallel inputs for data transfer.

The block diagram is shown in Fig. 15. This board contains eight modules of 1024 APEs working as a string of 8192 APEs. The string is divided into four channels of 2048 APEs, each channel being associated with an ADB buffer. Each channel is the equivalent of an ASPA board.

Each ADB buffer contains two sets of dual-port memory: the Input Data Buffer (IDB) and the Output Data Buffer (ODB). Each IDB and ODB plane is divided into two pages of 2K $\times$ 32 bits. For the IDB, one page is connected to two modules. For the ODB, one page is connected to the same two modules and the other page is tied to the VME interface. Such a structure allows the machine to be run in a full pipeline mode: front-end data may be written onto one page of each IDB buffer while other data are read from the other page by the modules; similarly, some results may be written by the modules onto one page of each ODB buffer while other results are collected from the other page via VME. However, it is also possible to keep an instance of the input data in the IDB until these data have been processed by the ASP. This allows the input data to be read back together with the ASP results for selected events. The control of the input data transfers is handled by the Input Data Buffer Address Logic (IDB-AL). When using the four front panel connectors, the protocol is asynchronous. Protocol errors and buffer overflows are detected. Transfers with an unequal number of words in each port are allowed. Similarly, the Output Data Buffer Address Logic (ODB-AL) allows transfers with an unequal number of words from the ASP to the four ODB buffers. This characteristic is of special interest for feature extraction algorithms.

With the exception that the HASPA board is not connected to the VSB bus, all the other elements (communications to the LAC and the VME, VDB debugging logic) are roughly similar to those of the ASPA board.

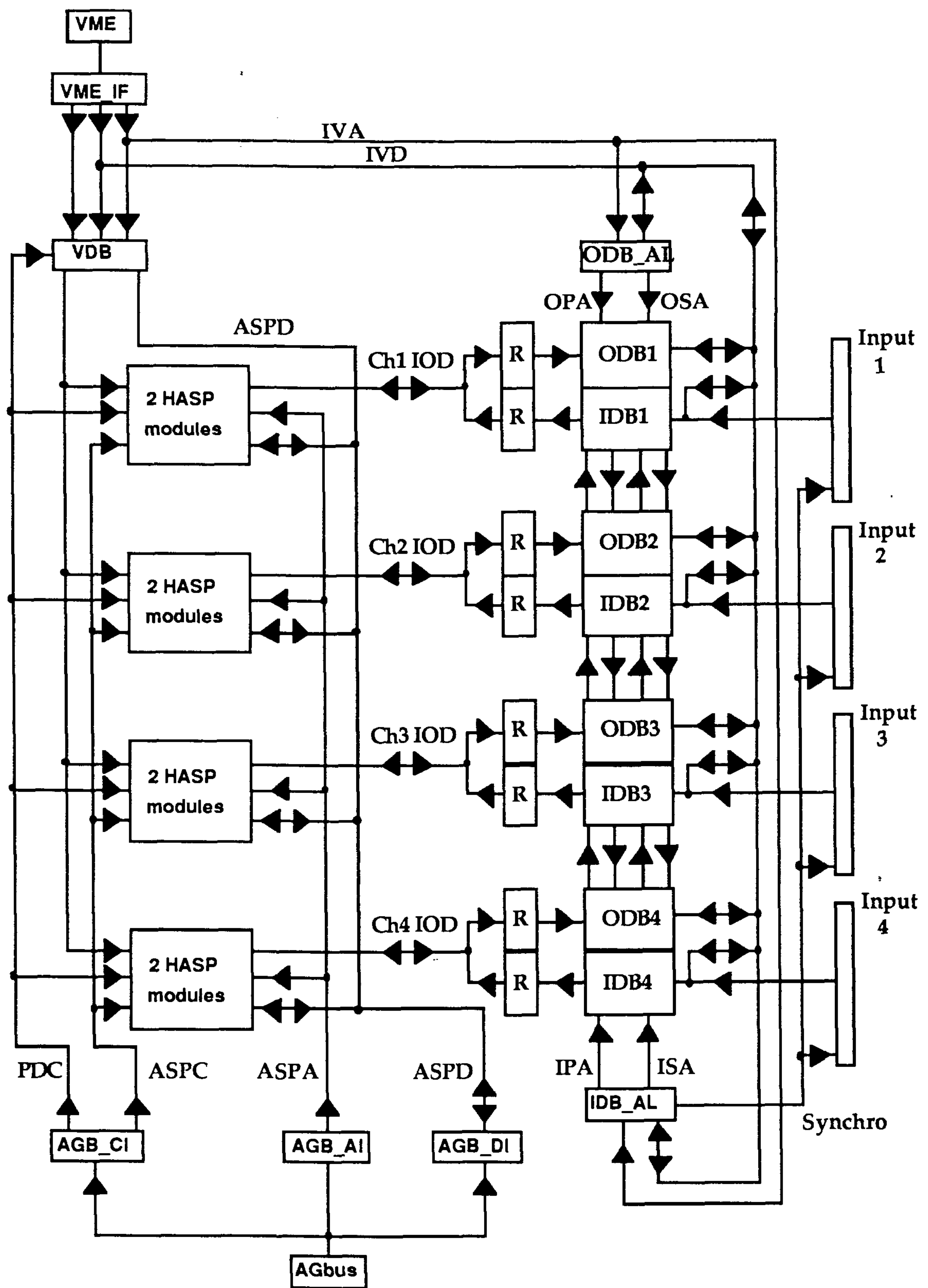


Figure 15 The HASPA card architecture

The HASPA card consists of two boards: a mother board and a daughter board covering about a third of the main board. Both are multilayer boards with six layers of signal and two layers for ground and Vcc planes. The daughter board contains essentially the direct input connectors, the differential receivers, and the input memory. The IDB-AL and ODB-AL are implemented in two 3090 (100 MHz) Xilinx PGAs.

Unfortunately, owing to the lack of hybrid modules, the two HASPA boards built have been only partially commissioned.

4. THE ASP EMBEDDED NODE (ASPEN)

With a view to using the ASP in real-time embedded applications, a software and hardware environment capable of meeting the requirements of future trigger and acquisition systems in terms of response time, scalar processing, and data I/O has been developed at Saclay. This development led to a new machine (ASPEN). In the ASPEN architecture [12], a high-performance, standard microprocessor acts as the ASP controller and scalar processor (see Fig. 16).

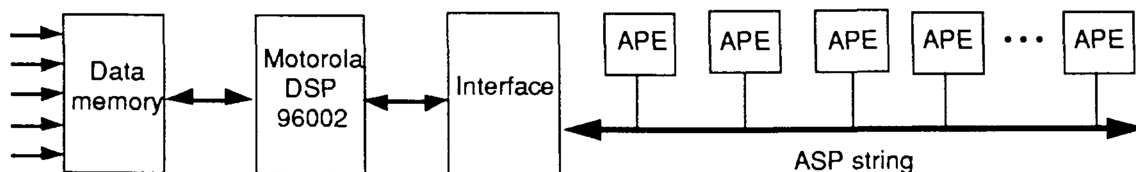


Figure 16 The ASPEN architecture

The ASP software execution is supervised by the microprocessor, and the control flow of scalar and vector operations is directed by the microprocessor program execution. The ASP string operates as a co-processor, and can be seen as a parallel-processing accelerator. The ASP program may run in fine synchronization mode, where it receives requests on an instruction-by-instruction basis, or can run in asynchronous mode to execute predefined block procedures. Special rendezvous synchronization points are automatically inserted by the software to exchange data and results between the microprocessor and the ASP. In addition, the ASP vector elements may be seen in the microprocessor memory space as normal variables that may be accessed sequentially or according to dynamic activity criteria.

In the first implementation of this architecture, the microprocessor is a high-end, dual-port, floating-point, digital signal processor (Motorola DSP96002). Pictures 4 and 5 show the ASPEN machine which is currently working at Saclay. The first external bus port is dedicated to the ASP interface, while the second port is left free for front-end data interfacing and dialogue with other parts of the system. Figure 17 shows the main elements of the interface.

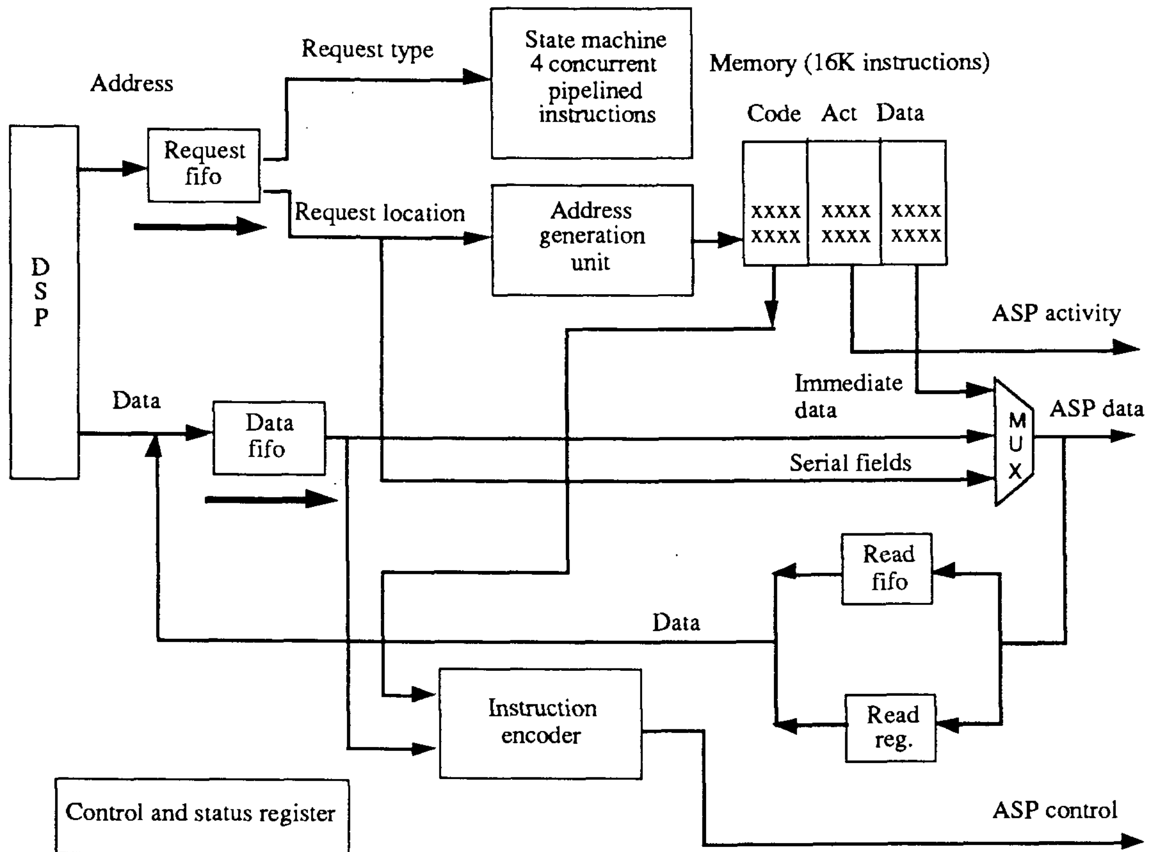


Figure 17 The ASPEN interface organization

A key element of this interface is a static memory which is loaded with the suboperation codes, the static operands and activity values, either for a single operation or for a block of operations. This memory is seen by the DSP through a fifo as a standard RAM. Addressing a location inside the memory will start the execution of a specific operation: the LSB part of the address will extract information from the memory while the MSB part will indicate the type of cycle to be executed.

The requests sent by the DSP through the fifo are pipelined by four state machines working in parallel in a five-step sequence (see Table 1, cycles m, m+1): Request detection (D), Fetch (F), Analysis (A), Execution (E) and Read (R). When the pipeline is established, the detection step occurs during the read step (Table 1 shows the pipeline progression for a typical sequence of five instructions: V, W, X, Y, Z). This structure maintains the rate of one request for each DSP cycle.

Table 1. The ASPEN pipeline structure of the request

Cycle	Pipeline level				
	D	F	A	E	R
m – 4	V	–	–	–	–
m – 3	W	V	–	–	–
m – 2	X	W	V	–	–
m – 1	Y	X	W	V	–
m	–	Y	X	W	V
m + 1	–	Z	Y	X	W
m + 2	–	–	Z	Y	X
m + 3	–	–	–	Z	Y
m + 4	–	–	–	–	Z

If dynamic scalar data is needed for an operation, it is provided on the DSP data bus through the data fifo. Results are written in the read fifo or directly on the B port of the DSP. A unique request from the DSP may start a block of operations in the ASP string. During the execution of a DSP request by the ASP, the DSP may continue to work unless it has to wait for the ASP result.

This first ASPEN system was simulated, built and successfully tested in a 512 APEs configuration. It has been under continuous operation since October 1992. Two generations of VASP-64 circuits have been used on this machine: ES2 CMOS circuits and Hughes SOS H1-B1 and H1-B2 versions, operating at up to 30 MHz instruction rate.

5. THE ASTRA MACHINE SOFTWARE

This section describes the status of the operating system, based on UNIX, and the programming environment developed for running an application on the ASTRA machine.

The target machine, as described in the previous sections, consists of several hardware modules. The three independent hardware activities, working jointly, give the idea to present the machine as having three intercommunicating levels: the HAC (High-Level ASP Controller) to refer to the Sun SPARCstation; the IAC (Intermediate-Level ASP Controller) to refer to the FIC8232 (68030 processor); and the LAC (Low-Level ASP Controller, see Section 3.3). The system and application software reflect the three-level hardware of the machine (see Fig. 18).

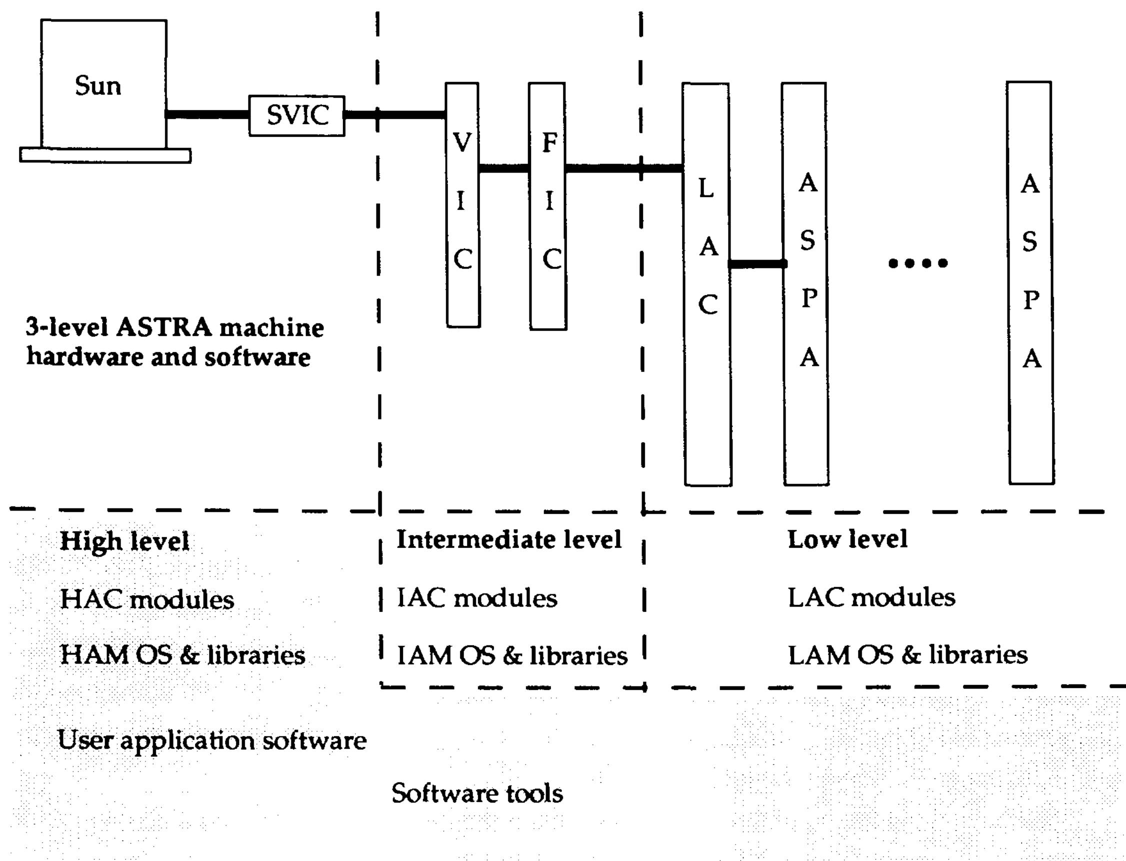


Figure 18 Three-level software for the CERN ASTRA machine

5.1 Writing an ASTRA application

The ASTRA programmer has to write applications with the three-level architecture of the machine in mind. An ASTRA application requests the programmer to support a software main module for each hardware level of the machine: he has to write the HAC, IAC and LAC modules defining the interfaces between each module. A number of facilities are available in order to write this kind of three-level application.

The main idea is to program each layer using services provided by the lower layer. That is, the HAC part will use the IAC procedures in remote to drive execution on the IAC, and the IAC part will use the LAC procedure in remote to drive execution on the LAC. The system so defined uses a Cross Procedure Call (or Remote Procedure Call) mechanism for control and data communication between each hardware level. The procedures defined in one module and called in a module of a different level are called cross-exported procedures. The way the cross procedure calls are executed is completely transparent to the application programmer.

5.2 Software development tools: the compilers

The language used for writing an ASTRA application is Modula-2 with some restriction for each level. Since the target machine consists of three different hardware modules, three different commercial compilers are used to generate the corresponding target executable code. The three compilers are:

gpm: Garden Points Modula-2 for Sun SPARCstation architecture [13];

ace: The ACE Cross Modula-2 Compiler for MOTOROLA 68030 architecture [14], [15];

lamc: The ASPEX Microsystems Ltd. Cross Compiler for Low-Level ASP Controller [16], [17].

In order to simplify the task of the programmer, a multi-level compiler generator is provided. The programmer has to write the definition and implementation modules for each level. Those modules will be compiled using a program called 'aspc' [18] which drives the execution of the right compiler according to the target level that the module represents. Furthermore, the aspc compiler will generate the necessary code for implementing the cross-exported procedures.

The compiled modules are linked to form a single application program ready to be executed on the three-level hardware components of the machine.

The linker program is called 'aspl'; the aspl links the modules taking into account the three different hardware targets.

5.3 Operating system tools and run-time libraries

The ASTRA operating system assists the user in running a three-level application.

The machine initialization, the target code downloading, the program execution and the cross-communication modules are the tasks of the ASTRA operating system.

A set of run-time libraries is available. These are provided for each level, for cross-procedure call management, for specific instrument resources interface, and for data communication.

These operating system tools are:

- LAC-OS
- IAC-OS
- HAC device driver
- SVIC-VIC link initialization tools
- LAC and APEA initialization tools (APEA is equivalent to ASPA)
- LAC and IAC downloading tools.

The run-time libraries are:

- Cross-procedure scheduling, polling and synchronization management
- Shared memory management between HAC and IAC
- LAC SDB fifos interface
- APEA ADBs interface
- APE debugging tools
- General-purpose LAM libraries.

5.4 ASP documentation

The documentation which is available for ASP allows any beginner to understand the basic concept of ASP, to use the ASP simulator, and to write application programs running on the ASTRA machine. The basic documentation was issued by ASPEX and a number of documents, written by CERN/MPPC, are available to help the non-specialist.

5.4.1 ASPEX documentation

ASPEX Microsystems Ltd. provides a set of basic ASP documentation for the ASTRA machine and for ASP simulation. These documents are:

- A User's Manual of the LAM Compiler [16]
- A LAM Programmer's Reference Manual [17]
- A User's Manual for the ASPC, Version 1.1 and ASPL [18]
- An ASTRA Application Programmer's Reference Manual [19]
- A User's Manual for the VASP-SIM Simulator [20]
- A VASP-SIM Procedure Library User's Manual [21]
- A VASP-SIM Procedure Library User's Manual of the Advanced Arithmetic Package [22].

5.4.2 CERN documentation

In order to help and assist the programmer working on the CERN ASTRA machine, additional documents have been issued by the MPPC Collaboration at CERN. They complete the original ASPEX documentation and are very useful for ASP programming beginners:

- The 'ASP Cook Book' which was written for the ASP beginner in order to present in a simple didactic way the principles of the ASP concept [23];

- The 'ASPA Programming User Guide' which shows in full detail how to start programming an application on the CERN ASTRA machine [24];
- The 'ASPA Installation Guide' presents all the relevant information about the installation of the hardware and software tools of the CERN ASTRA machine and its environment [25].

5.5 Graphics tools

A dedicated graphics interface has been developed at CERN in order to provide a self-explanatory output of the debugging facilities: the contents of each APE can be dumped and visualized in a Sun window at any step of the program. At the moment, it only works when using the simulator installed on the Sun. The implementation of the graphics interface for the ASTRA machine will be a 1993 upgrade of the system. A Modula-2 graphics interface has been implemented for GPM using the OpenWindows 3.0 graphics environment at the HAC level. All these graphics facilities are documented in 'Graphic tools for the ASTRA machine' [26].

All the relevant ASP-CERN documents have been regrouped in the open documentation which is available for the ASTRA users: the CERN-ASPA Machine User's Book [27].

Another important programming help facility was created in 1993 by ASPEX: the ASTRA (ASP System Test-bed for Research and Applications) debugging team which provides the ASTRA programming user with a fast debugging feedback loop [28].

5.6 Other ASTRA programming methods

Two other solutions have been developed for programming the ASTRA machine. In particular, when the ASPA and LAC cards were delivered at CERN, a simple two-level ASTRA-OS was developed using only HAC and LAC, for Modula-2 and C interfaces. This system uses a programming approach similar to the standard three-level operating system. As this system implies a lot of communications between the Sun and the LAC through the S-VIC, producing heavy overheads, it was only considered for development of simple offline applications.

Another system has been developed by LAL-Orsay for a specific, real-time application (see Section 7.1.1). This system puts emphasis on the IAC which is in charge of data I/O. The IAC works under the OS-9 operating system, and the code was written in the C programming language. It makes use of the low-level compiler (lamc) from ASPEX.

5.7 An example of applications programming on the ASTRA machines

Two introductory examples for the CERN ASTRA machine are given in the Appendix. Section A.1 gives the basic skeleton of an ASTRA program and Section A.2 shows an example of a complete case-study application program.

5.7 An example of applications programming on the ASTRA machines

Two introductory examples for the CERN ASTRA machine are given in the Appendix. Section A.1 gives the basic skeleton of an ASTRA program and Section A.2 shows an example of a complete case-study application program.

6. A CCD INTERFACE TO ASTRA

In order to be able to test the processing power of the ASP, a large amount of data was require. A CCD camera was found as a rather simple device able to generate this large data file at a suitable rate.

A first VME prototype of a single direct output, transferring the CCD data (pixel by pixel) at a readout speed of 100 ns per pixel, was constructed at CERN. The interface is flexible enough to support various types of CCDs (full-frame or image-memory transfer), even with a different number of pixels for lines and columns.

Besides applications in HEP, the CCD camera linked to the ASTRA machine was used as a data source for testing iconic processing algorithms and as a demonstration platform for the ASTRA machine.

6.1 Description of the CCD readout and its interface to the ASTRA machine

The card is plugged into a VME slot of the ASTRA machine and is controlled by the CPU master of the crate or by the Sun host through a SVIC-VMV-VIC interface (32 bits). Different CCD targets from Thomson-TMS [29] were used: the TH7863 (image-memory CCD; 384×288 pixels), the TH7883 (full-frame CCD; 384×576 pixels) and the TH7895 (two outputs). The configuration of the CCD data acquisition system built for these CCDs is shown in Fig. 19.

The lines are sequentially transferred to the output register which is read pixel by pixel at the output diffusion. The sequential readout time at 10 MHz is 15 ms for the TH7863 and 30 ms for the TH7883. A pixel cycle is divided into three time slots of 33 ns each: reset, floating, signal. With the double-correlated sampling the analog value of the pixel is stable for ~ 80 ns for 8-bit digitization. The memory is organized in four benches of 64 Kbyte sequentially loaded by the digitized value of successive pixels. It can be read from the VME in 32 bits representing four adjacent pixels.

CCD interfaces were built around a 4005 Xilinx FPGA. Implementation of the hardware in Xilinx FPGAs allows the same card to be configured for full-frame or image-memory CCDs. The card can also be programmed to be used with the HASPA high-speed interface. Free-run or one-shot mode are available for image-memory CCDs.

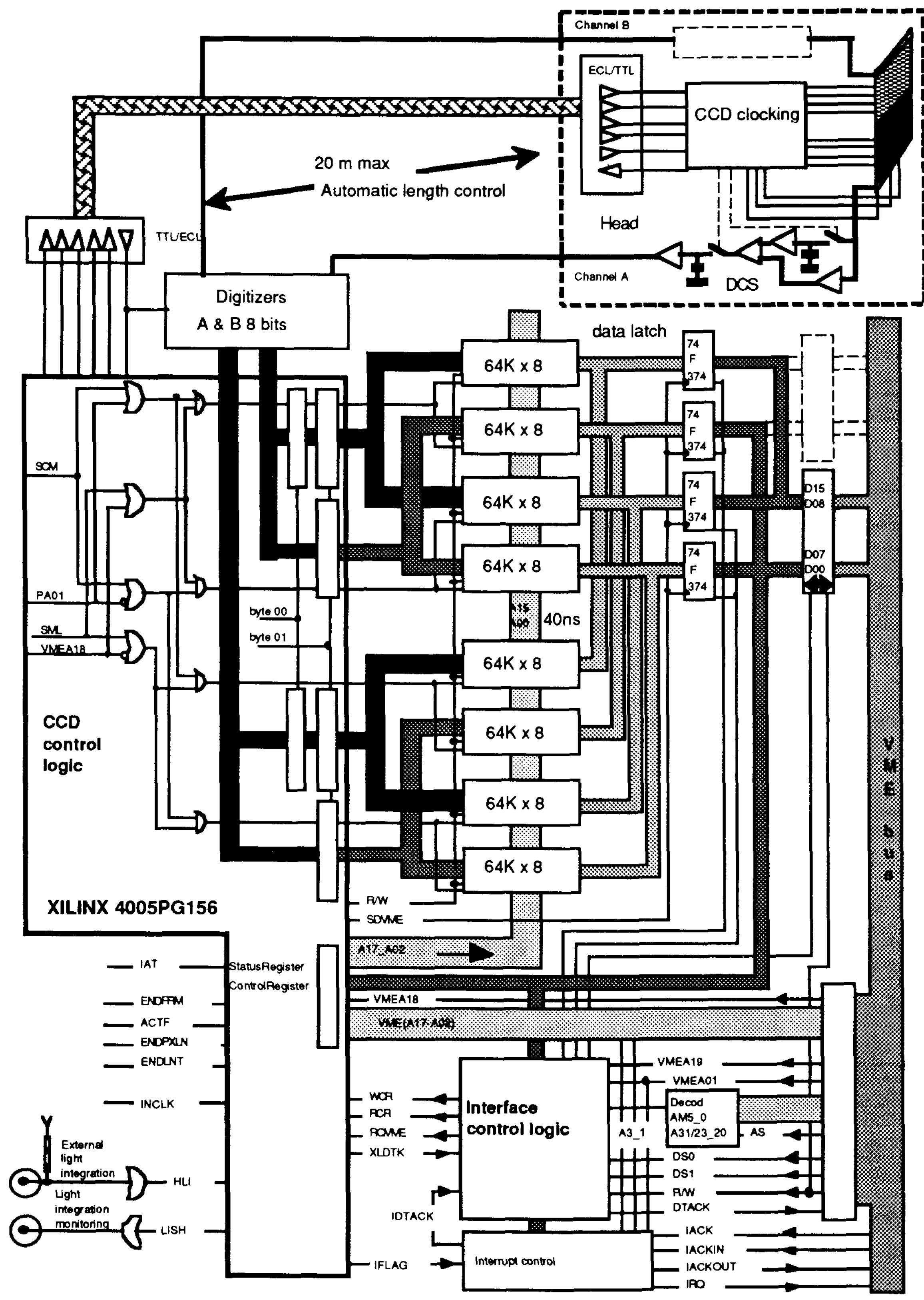


Figure 19 The full CCD acquisition system

The interface has been enhanced with 512 Kbyte of memory and two analog channels to work with a 512 × 512 pixel, two-output CCD (Thomson TH7895).

A printed circuit board has been designed and is populated with 16 special memory units giving a total of 1088 Kbyte to work with the 1024 × 1024 pixels CCD (TH7896 from Thomson) in two-channel mode. The 10-bit digitization, the on-line pixel correction, and the VSB interface are not yet implemented.

6.2 Test results of the CCD-ASTRA system

The TH7863 CCD linked to the ASTRA machine has been running for a test-bench demonstration. For this purpose, the image as seen by the CCD was stored in the VME memory area and mapped inside the ASP in 2K patches through the DMA channel (8 bits per pixel loaded in each PE). As an example of feature extraction, the patch was then processed using morphological operators for contour outline (edge detection) of the image. For this algorithm, the measured processing time in the ASTRA was 24 μs per patch and is independent of the size of the patch. For this application, the image processing at video rate is clearly achieved.

6.3 A CCD interface for the MPPC machine

The main interest of the 16K MPPC machine would have been its ability to run complicated algorithms in parallel on a large amount of data, generating a huge processing power. In parallel to the development of the HASPA card a study has been carried out for realizing the CCD–HASPA interface.

The design of the interface between the data source and the machine uses the full parallel I/O capability of the HASPA card. For this purpose, the data are presented in words of 32 bits. Figure 20 gives a block diagram of the interface between the CCD VME card and a two-HASPA-card machine.

The FPGA programming allows various patterns of rectangular patches smaller than or equal to 16K pixels to be processed. The transfer time, including all overheads mainly due to the patch pattern preparation in the FPGA, must be smaller than the image patch processing time. The 32 fifos of 512 bytes can be loaded in less than 250 μs, the transfer from fifos to APEs needs 50 μs, and the process time for a typical image was estimated by simulation to be ~ 1 ms.

With the 16K machine, a 256K pixel frame could be processed in 16 ms, which is fast enough for handling online, video-rate image processing.

It is worth pointing out again that the processing time does not depend on the image size, provided the number of APEs be enough to accommodate all the pixels.

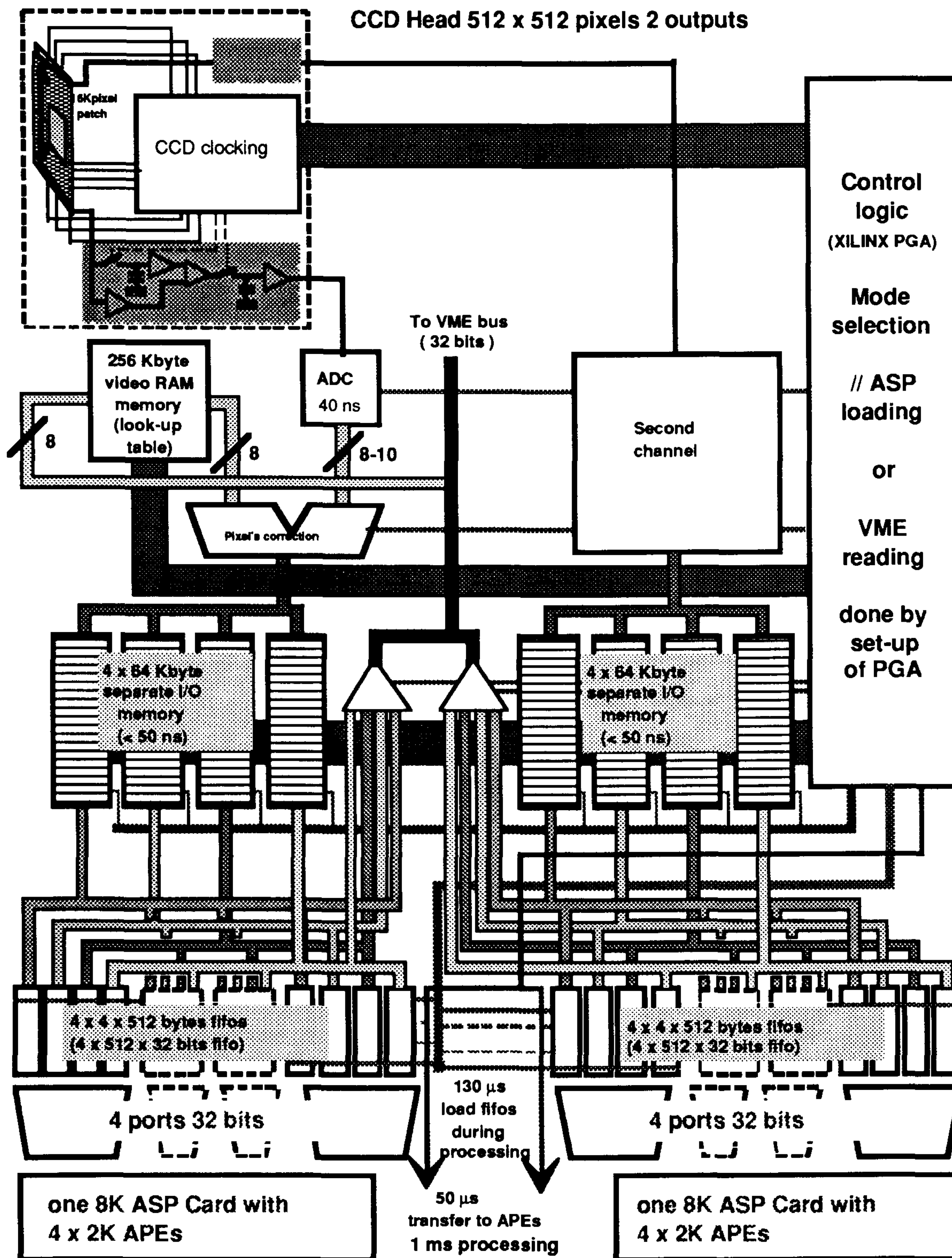


Figure 20 The two-output CCD (512 × 512 pixels) interface for the 16K MPPC machine

7. APPLICATIONS

During the course of the MPPC project, all partners studied, according to their individual motivation, a large variety of ASP applications. Almost all of these applications arise in the context of preparatory work for the future Large Hadron Collider for triggering physics events.

The generally accepted trigger structure, borne out by physics simulations, is the following: after a reduction of the initial event rate of 40 MHz by a custom-made, possibly analog, first-level trigger, relying on calorimeter windows and muon identification, the event passing rate is of the order of ~ 100 kHz. At this rate, reduction 'algorithms' of some complexity will be required to reduce rates further. They are based on data from the first-level trigger and on additional local detector data that may be extracted from some buffering device or intercepted 'on the fly' as data pass from one system part to the next, e.g. as they are pushed from geographically spread front-end pipelines to data concentrators.

At the beginning of the project, before the first ASP compact machine became available, the only way of studying parallel algorithms was to run simulations on a VASP simulator supplied by ASPEX.

7.1 Application studies on a VASP simulator

Timing results given by the VASP simulator working on a Sun station are based on the assumption that the ASP machine could work with a 25 ns time slot for sub-instructions.

7.1.1 Tracking and calorimetry for the SDC level-2 trigger

The first study of the second-level trigger of the SDC detector was made at Saclay on the VASP simulator to get an idea of the feasibility. The architecture of the trigger was later updated. However, this study gave a lot of information about the timing and the way to program this kind of algorithm.

A complete program was developed and run on the VASP simulator for an architecture based on the idea of two ASP machines working in parallel: the first one analyses the data coming from the tracking, and the second one the data coming from the calorimetry, each detector using its own granularity (see Fig. 21). The decision is taken at the end by comparing the transverse momentum from the tracking with the energy from the calorimetry.

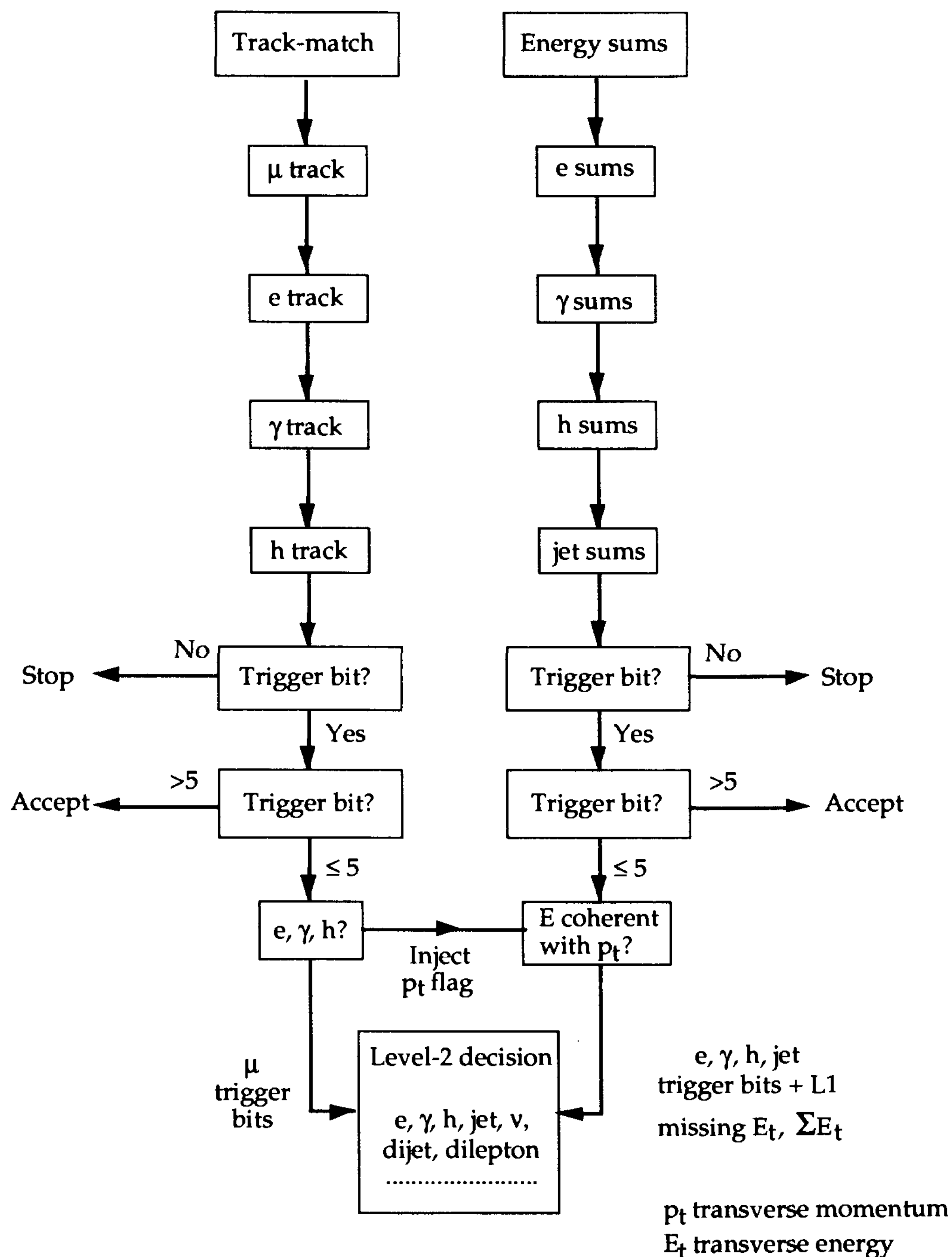


Figure 21 The SDC level-2 tracking and calorimetry trigger

The tracking algorithms should identify muons, electrons, photons and hadrons using the tracks in the detectors: this confirms or not the results of the level-1 trigger. The event will be killed if no particles are detected, stored if the number of particles is greater than five, otherwise, for a number of particles from one to five, a comparison is done with the calorimetry results.

The analysis of the energy deposited in the various parts of the calorimeter allows the identification of electrons, photons, hadrons, or jets. This is followed by a cut on the energy level of the identified particle. Each particle is treated by an algorithm using summation of energy in the neighbouring region. The final decision depends on the number of particles identified.

All these algorithms have been tested on the VASP simulator. The processing time for the different configurations has been estimated. For example, the determination of two electrons needs 22 μs, 1 gamma + 1 hadron needs 29 μs; a library of procedures useful for the algorithms allows more tests and investigations to be done easily.

These results were found to be very encouraging and new studies have been started to improve performances (see Section 7.2.2).

7.1.2 The LHC high-transverse-momentum muon second-level trigger

At the Large Hadron Collider (LHC) the high luminosity (4×10^{34} cm^{-2} s^{-1}) will create a severe environment for the detectors, making physics discoveries a difficult challenge.

The most promising particles for TeV physics, which could be used to trigger interesting events, are the relatively rare high p_t (transverse-momentum) muons which remain after filtering out all the other unwanted charged particles. A first challenging goal is to discover the Higgs particle through its decay into four muons.

In the most optimistic phase of the first-generation detector design, accepting Monte Carlo simulation predictions at face value, the general opinion tends to minimize the need for a second-level trigger decision. The main argument put forward is as follows: using a simple first-level decision on the p_t muon threshold adjusted at a sufficiently high value, the trigger rate could be tuned down directly to the value required by the level-3 trigger (kHz range).

However, physicists must be prepared for many eventualities: the background rates may be underestimated, or one needs for the physics search to run p_t muons at a lower threshold, or the binary yes/no p_t muon information coming from the first-level trigger is not sufficient as a signature for the high p_t muons, etc. This is why a detector model (see Fig. 22), optimized for a second-level muon trigger for LHC [30], was studied and simulated using the ASP architecture. Owing to its inherent parallelism, the proposed algorithm automatically provides multimuon information if there is any, assigning to each muon the charge sign and momentum.

The proposed second-level-trigger ASP procedure has three phases: the loading (detector mapping into the ASP), the preprocessing (master-point determination of the muon hits) and the tracking (based on a p_t dependent track-code search algorithm).

In the simulation we assumed five superlayers with four detector layers in each [Ref. 3, p. 69]. In the loading phase, providing a parallel feed into the ASP substrings, one can perform within 5 μs the 'iconic bit-mapping' of wire hits to the memory cells of the associative processing elements of the string.

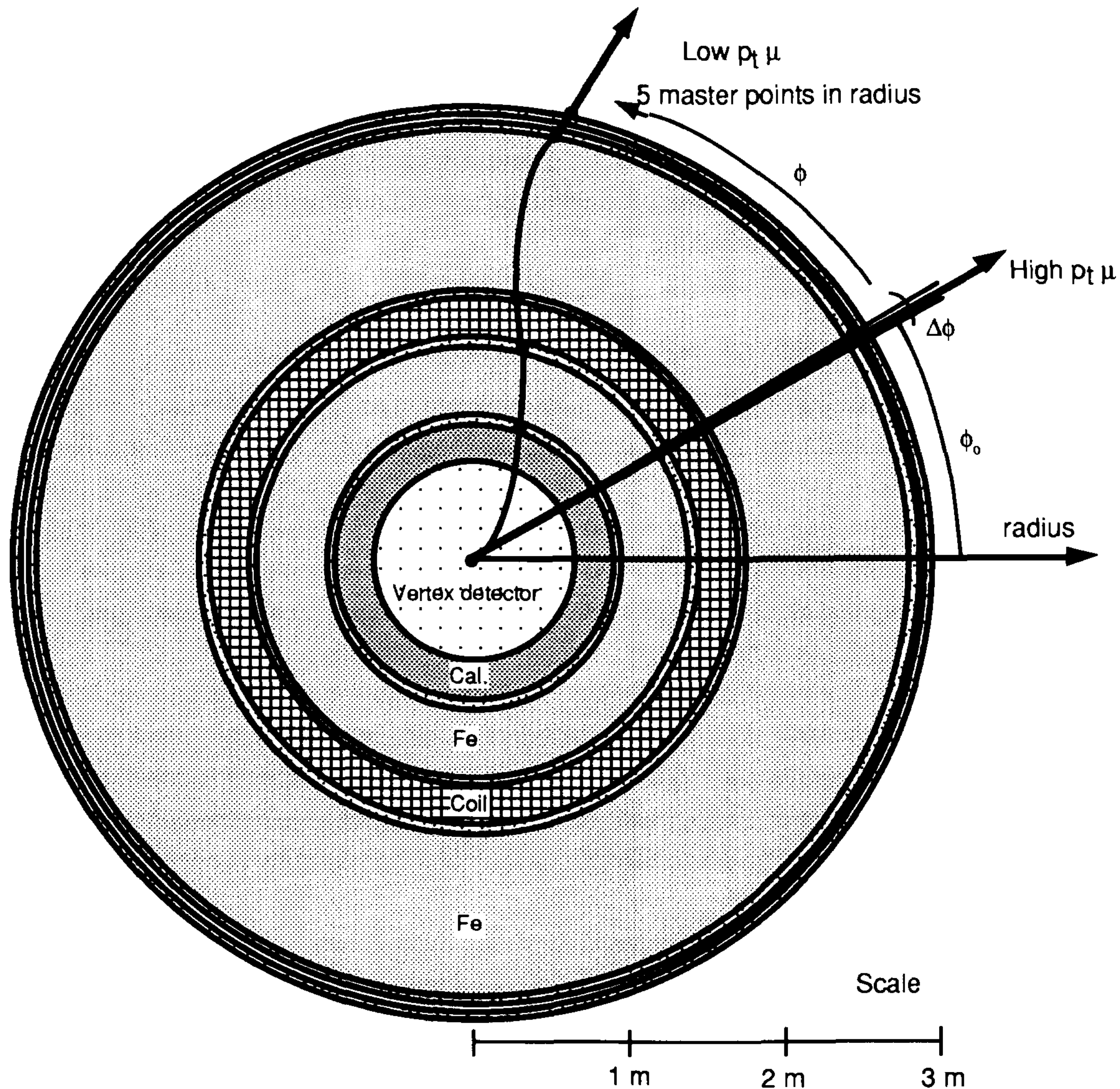
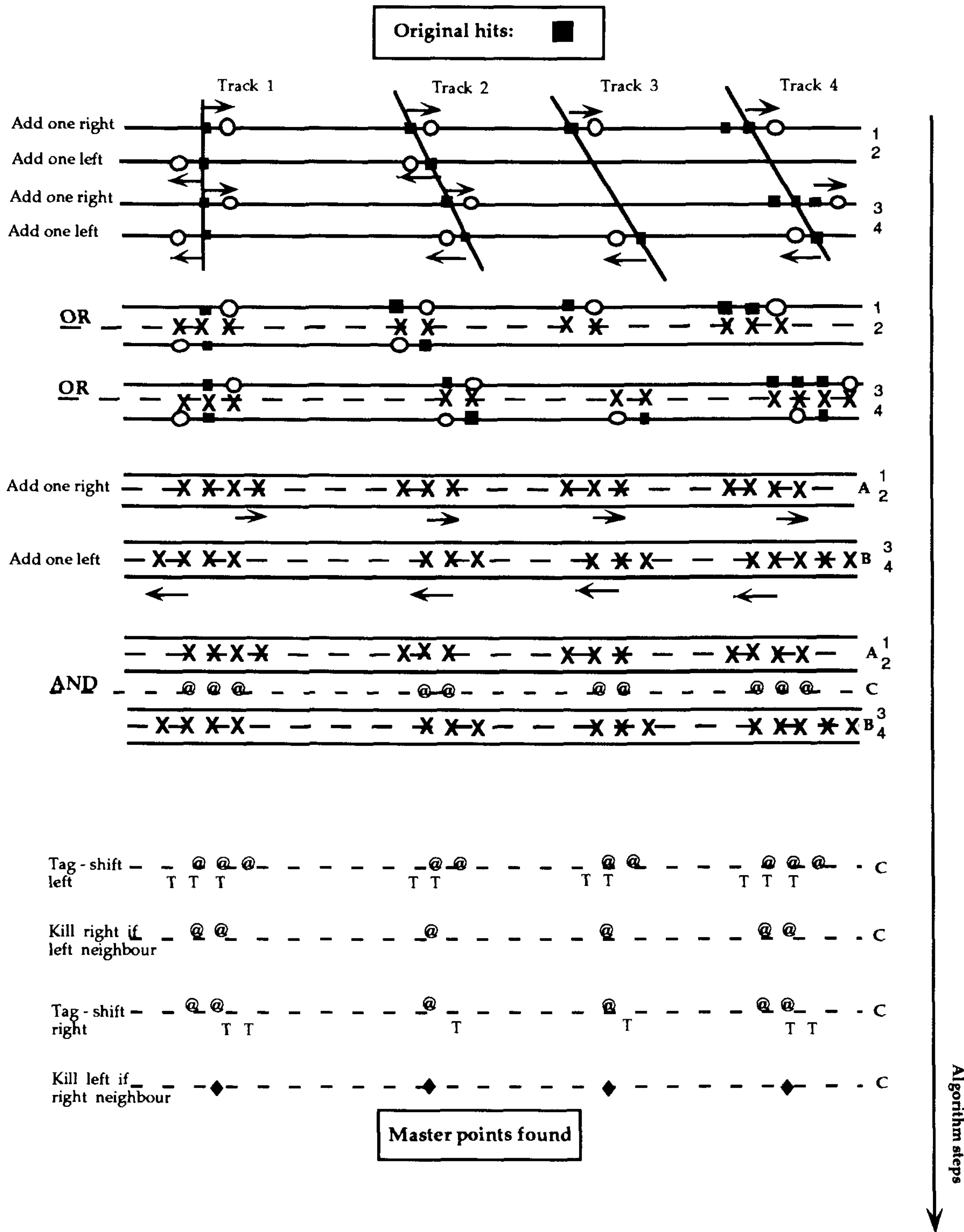


Figure 22 The LHC muon tracking modelling

In the preprocessing phase one performs an 'iconic averaging' of hits in order to calculate the master-point coordinates (see Fig. 23). This algorithm takes into account the chamber inefficiency (no hits in some layers) and the possibility of multiple hits in the same detector layer (the passage of a given particle can produce double or even multiple hits).



Track 1: Ideal straight track, full efficiency
Track 2: Inclined track, full efficiency
Track 3: Inclined track with 2 missing hits
Track 4: Inclined track with multi-hits and a missing hit

Figure 23 The 'iconic average' algorithm, illustrated for typical muon track signals

The overall efficiency of this algorithm to get all five master-points belonging to a given track is 97.5%, assuming a realistic chamber efficiency of 95%. The processing time is close to 5 μs.

The tracking phase provides the sign and a momentum estimate for all the master-point sets which coincide with any of the predetermined p_t muon track-code combinations. The search for all particles is performed in parallel along the track-code list. The reference ordered list, in decreasing p_t values, of valid track-codes is precalculated by Monte Carlo simulation.

The number of valid track-codes above a given threshold depends on the ϕ coordinate binning. The effective Δ–ϕ granularity of the 'iconic mapping' is dictated by the size of the multiple Coulomb-scattering at the selected momentum threshold. Accepting this value as a guideline one can limit the necessary number of track-codes to below 100, which ensures the execution time within 10 μs.

In most of the first-level triggers there will be no match, thus the procedure is finished in total within 20 μs. For the lucky cases, of course, the third-level processing will start. Part of this subsequent procedure—multiplicity, muon sign analysis, etc. can already be executed in the ASP itself by correlating the successful track-code matches.

7.1.3 The transition radiation tracking (TRT) detector for LHC. Simulations for applications of ASP modules for a 100 kHz trigger

This study arose in the context of the preparatory work for LHC and was done as part of the RD11 (EAST) project, as one possible implementation of second-level triggers [31].

In this architecture, it is assumed that first-level trigger results and additional local detector data are pushed from selected regions of interest of the detector space to the processing machine under assesment through dedicated data routers. External hardware, namely local buffers and the first-level trigger, sends control signals and local data, selected in a region of interest (RoI), into the architecture in question.

This is usually described as a 'push' architecture, as opposed to a readout in which the device executing the algorithm also manages the readout ('pull' architecture).

7.1.3.1 Feature extraction

The investigations on the basis of the ASP architecture were limited to the 'feature extraction' part of these algorithms, i.e. to the task of converting raw front-end-formatted data to quantities ('features') meaningful from the physics point of view. Features are interesting in restricted areas of individual subdetectors only, and the task of feature extraction is not unlike image processing tasks. Feature extraction can in a natural way make use of multiple devices operating in parallel, on different RoIs, and for different subdetectors.

The work on ASP was part of a systematic study of possible architectural solutions for two feature-extraction tasks. These were defined as characteristic second-level trigger tasks in terms of physics goals, detectors, and triggering algorithms, and with fixed assumptions about detector and first-level trigger electronics. Benchmark implementations of such algorithms were done on seven competing architectures, on available hardware where possible, by detailed simulation otherwise. Conclusions about the ASP must be seen in the context of this comparative study.

Both algorithm definitions were the result of close collaboration with R&D projects (RD1 and RD6) pursuing the corresponding detector developments. The algorithms, or rather the problems together with a possible algorithmic solution, are defined in internal EAST notes [32]. The pilot tasks and algorithms reflect a certain state of detector development, frozen for our purpose. Both algorithms have the objective of optimizing the retention of electrons (signal) and the rejection of QCD jets or other phenomena misjudged in the first-level trigger (background).

7.1.3.2 Benchmark definition

The physics features extracted from fine-grain local data in multiple detector windows have to be combined later into global decisions by correlating data from different subdetectors and RoIs. This global decision has not been addressed with ASP in mind, as a different parallelism will be at work, and high-level programmability will be postulated in order to evolve algorithms reliably with the improved physics and detector understanding.

Feature extraction will also have to be preceeded by a 'router', which extracts the RoI data and formats them for the feature extraction device. The embedding is shown in Fig. 24.

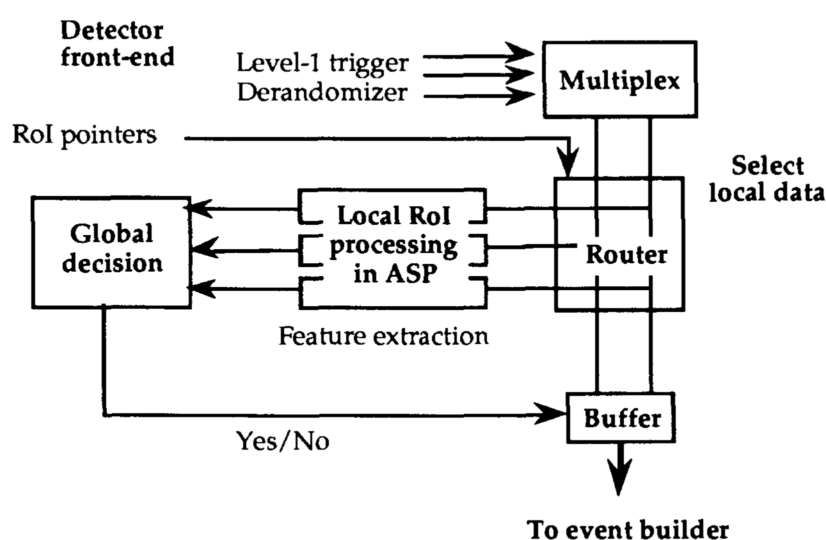


Figure 24 The router

Very briefly, the pilot tasks are as follows:

- a calorimeter algorithm for a RoI of 16×16 towers. Each tower of our calorimeter is subdivided into 2×2 electromagnetic cells, one hadronic and one mixed (wedge-shaped) cell. Each cell is defined by its energy, a quantity known to a precision of 12 or more bits (gray value). In our simulation the RoI of 256 towers corresponds to $\Delta\eta \times \Delta\phi = 0.5 \times 0.5$, in physical space. The objective of the algorithm is to find features, i.e. decision variables suitable for distinguishing electrons from pions or from hadronic jets, or even pions from jets. Inside the RoI, a near-circular region has first to be defined in which the peak energy deposition is fully contained (cluster area). The second moment of the cluster radius in two dimensions, or energy sums in different ring-shaped zones and longitudinal volumes of fine granularity, have to be calculated over the cluster area since these variables best distinguish electrons from other background phenomena. They are likely to contain all relevant information for electron/pion/jet discrimination that cannot be explored in a first-level algorithm. They also allow improvement of the positional resolution inside the RoI.
- a tracking algorithm based on a straw geometry as pursued in the RD6 project (transition radiation tracker or TRT), operating in a magnetic field. In the projection natural for the TRT, i.e. in the z/ϕ plane, which corresponds closely to the readout coordinates straw/plane number, tracks appear as straight lines, with the slope dϕ/dz corresponding to p_t, the position along ϕ indicating azimuth, and start and end point crudely indicative for z.

The input data appear as an 80×240 image, with each pixel taking the value 0, 1, or 3. The algorithm then recognizes patterns of digitizing that correspond to high-momentum tracks, taking into account the pulse-height distribution of digitizing for identification of electrons. The algorithm consists either of making histograms along z (the simplest being without a result for p_t), or along roads of different dϕ/dz (with an indication of p_t).

For both pilot tasks, the algorithms were simple enough to be reprogrammed or even hardware-implemented depending on the architecture to be studied. For a clear definition, algorithms were given in a high-level language, but some freedom was left so that equivalent but different algorithms were considered acceptable. Data sets for testing were provided. They contained data values inside RoIs, for signal and background collisions, at different levels of luminosity (minimum-bias background).

7.1.3.3 Benchmark results

The benchmark results were published in Ref. [33]. The comparisons as presented in the following tabular form are, of course, an oversimplification. More detailed discussions than can be exposed here are given in several internal EAST notes [34]. The ASP results were obtained using the proprietary ASP simulator, assuming a clock

frequency of 20 MHz. The ASP execution times (see Table 2) came out best amongst the SIMD machines studied, but did not quite match the expected 100 kHz rate. Several pipeline implementations did, however, achieve this goal.

Table 2. Benchmark results

Architecture / algorithm studied	Measured (best possible) execution time (μs)	Latency estimate (μs)	Comment
ASP/TRT	33.5 (9.5)	50.0 (25.5)	10 (2) systems of 2048 APEs each
ASP/Calorimeter	20.6	Not available	5 systems of ≤ 2048 APEs

7.1.3.4 Discussion of the ASP implementation

For the TRT algorithm, local pre-histogramming along z (a look-up table reducing every 16 bits along z into a 4-bit count) was assumed to be done outside the ASP. The information loss related to this packing operation is local, and does not preclude the subsequent histogramming at different slopes $d\phi/dz$. Apart from increased time for data transmission, histogramming with unpacked data increases the execution time to 46 μs. For the calorimeter, the algorithm implemented uses different test statistics from most other implementations, but has been shown to be qualitatively equivalent. It is based on applying thresholds to six different energy sums (central peak, near-neighbourhood ring, wider neighbourhood for both electromagnetic and hadronic energy), i.e. on simple convolutions. The widest areas considered are 6×6 towers; the window slides dynamically over all possible positions in the RoI. The resolution for some energies is refined to the electromagnetic tower size, i.e. a fourfold number of cells is considered.

Implementation details: different slopes were done in parallel, in independent ASP substrings. This assumes a parallel data copy into several identical ASP systems of 2048 APEs each. For the calorimeter, the six sums (= convolutions) are done in parallel in five independent ASP strings, with different precision used in the sums. The total number of APEs is 6400, the largest substring is 2048. Here, too, data have to be replicated into multiple strings/APEs, to achieve efficient execution.

The timings given are for the board design of MPPC, ignoring the data duplication and the input bottleneck related to the existence of a single 32-bit-25 MHz channel for 2048 processors. The 'best time' (TRT) assumes only very crude histogramming along z, or tracks that span no more than three bins in ϕ. For the calorimeter algorithm, the logical connection between the results of the individual convolutions is not included in the timing. A separate timing has been performed for a TRT algorithm including the router

function. The router is a data reformatting unit necessary to present to the second-level trigger architecture a RoI in a suitable and invariable data order, independent of the modularity and readout order of the detector front-end electronics. In the present TRT prototype, the router also reduces the information from three bunch crossings into a single two-bit signal for each straw. The additional time for logical connection of time slices and data interleaving is 5 μs. The factor of 3 in required bandwidth would, obviously, constitute an increased I/O challenge to the ASP design.

In contrast with the conventional benchmarking of mainframe computer systems, which typically use very large application programs in a high-level language ('dusty decks'), compiler performance or adherence to portability standards was not part of our evaluation criteria. The main criteria for this real-time benchmark were:

- algorithm execution time, separated into the two aspects decision time (the time interval between successive decisions) and latency (time interval for a given event between start of data input and output of results): this assesses the overall feasibility of a given architecture to contribute as a second-level trigger device. Target numbers are 100 kHz for frequency, and of the order of 1 ms for the maximum latency of the entire level-2 system;
- practical solutions to the high bandwidth input: this addresses a typical bottleneck for many commercial systems which are targeted at compute-intensive problems, and also challenges the flexibility of architectures or their manufacturers in interfacing to specific user constraints;
- possible constraints on the order of input data; this aspect is relevant as we deal with architectures that typically achieve performance by high parallelism with distributed memory, or by pipelining data in a certain sequence. We assess here how much of the data selection (the router is shown in Fig. 24) has to be loaded with tasks that in a general-purpose device would be part of the algorithm itself;
- interfacing to a high-level decision-making unit and to the (physicist) user. The critical parameters in assessing the embedding difficulties of an architecture are flexibility with respect to algorithm parametrization, and the hardware possibilities of passing results (physics features) to a global device for overall decision making.

For this particular application, the ASP implementation has, in fact, put quite severe constraints on the order of data, has left the input bandwidth and the data replication problems unsolved in its existing hardware, and has not addressed 100 kHz communication of features with a workstation-type processor (64-byte packets).

7.1.4 Image coding applications

This particular application of ASP was studied at EPFL in the Laboratoire de Traitement des Signaux, led by Professor M. Kunt.

The recent emergence of various visual information processing applications, such as HDTV, videotelephone, videoconference, medical imaging or archiving, has led to increasing interest in image coding and image sequence coding techniques. In order to provide a more efficient representation of the visual data, redundancies need to be reduced. Several techniques achieving a high compression ratio while preserving very good image quality have been developed. Another important feature of these techniques is their complexity. For digital image storage applications, e.g. on CD-ROM, very fast encoding and decoding of pictures is desirable. In video coding applications, e.g. digital TV and HDTV, and videoconferencing, the requirement is even stronger; the encoding and decoding process should be performed at video rate (typically 50 Hz). As the image coding algorithms are highly parallel in nature, massively parallel implementation is a promising approach.

Two different techniques for image compression are described: a Gabor-like wavelet technique and an autoassociative neural network. Parallel algorithms are studied and implemented on the ASP, and simulation results are presented.

7.1.4.1 Image compression based on a Gabor-like wavelet transform

The subband- and wavelet-based techniques have shown their efficiency to reduce spatial redundancies in image compression applications. The Gabor-like wavelet transform performs octave band partitioning of the spatial-frequency domain. The Gabor filters have an optimal localization in the joint spatial/spatial-frequency domains, reaching the lower bound of the Heisenberg uncertainty relation [35]. The use of the Gabor transform is also motivated by the human visual system [36].

The coefficients of the Gabor transform are generated by a tree structure and rectangular, separable, bi-orthogonal wavelet transform [37]. Figure 25 illustrates the forward transform, and Fig. 26 shows an image and its corresponding transformed coefficients.

The algorithm requires convolution operations, which can be represented in the following form:

$$x(n,m) = \sum_{k=0}^{K-1} s(k) \cdot f_n(2m-k)$$

with $n = 1,2$ and $m = 0,1,\ldots,(K/2)-1$. In this equation, $x(.)$ represents the output of the subsampling blocks, m is the spatial location index, n is the frequency location index or the channel index, $f_n(.)$ is the impulse response of the filter $F_n(.)$, $s(.)$ is the input discrete signal, and K denotes the number of samples.

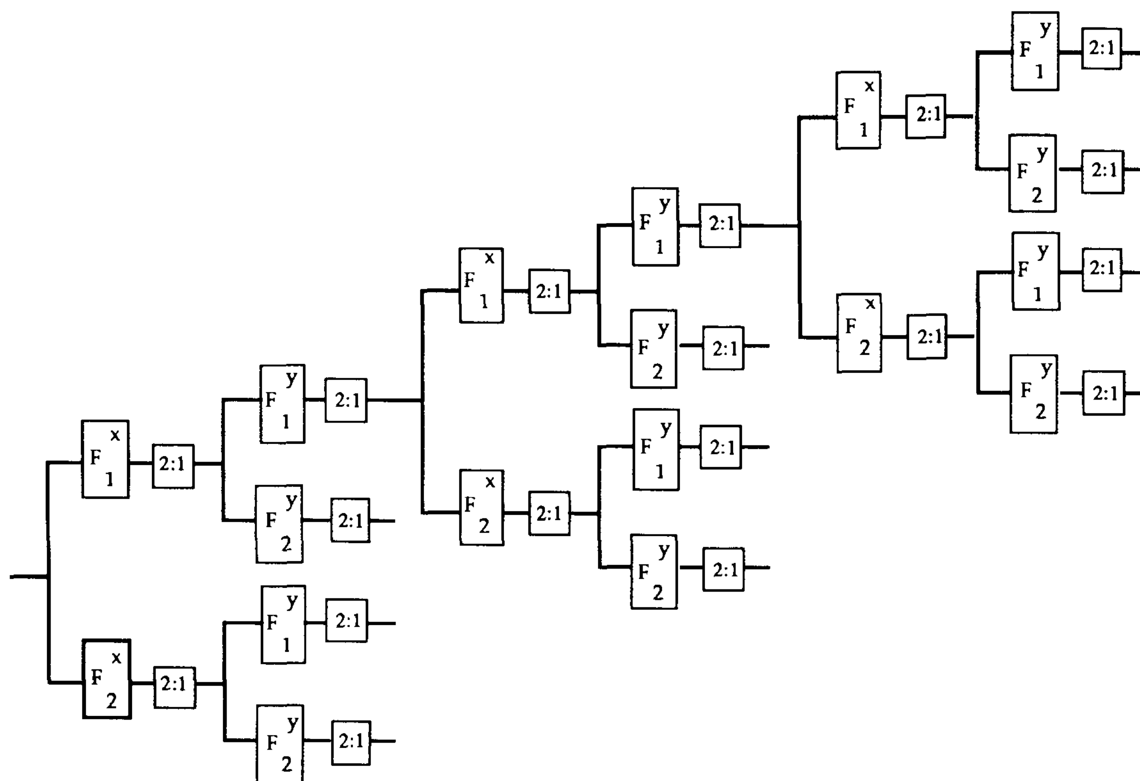


Figure 25 The tree-structured pyramidal Gabor filter bank

(a)

(b)

Figure 26 The Lena test image:
(a) original image, (b) its corresponding Gabor coefficients.

An alternative equivalent formulation for this equation using matrix multiplications is also possible:

$$\vec{x} = F \cdot \vec{s}\,.$$

In both cases, the problem is very suitable for massively parallel processing. A parallel algorithm has been developed and implemented on the ASP for matrix multiplication based on the outer-product algorithm [38], [39]. If we consider two square matrices $\boldsymbol{A} = (a_{ij})$ and $\boldsymbol{B} = (b_{ij})$, where $i,j=1,2,\ldots,n$, and the product $\boldsymbol{C} = (c_{ij}) = \boldsymbol{A}\cdot\boldsymbol{B}$, then the outer-product algorithm consists of swapping the outer two loops, which embody the parallelism of the algorithm, with the inner loop, as expressed by:

$$\boldsymbol{C} = \sum_{k=1}^{n} \begin{pmatrix} a_{1k} \\ a_{2k} \\ \vdots \\ a_{nk} \end{pmatrix} \left(b_{k1} \cdots b_{kn}\right)$$

Assuming n^2 processors are available, the kth column of the matrix $\boldsymbol{A}$, and the kth row of the matrix $\boldsymbol{B}$ are written in all the processors, and multiplications and accumulations are performed in parallel in each processor. A parallelism of n^2 is achieved for the multiplications and additions, resulting in n iterations to complete a matrix multiplication. On the other hand, by assuming a sequential I/O, the writing and reading of each element of the matrices leads to a n^2 dependency for the computation time [38]. Since the present ASP library supports only integer arithmetic, the matrix elements are quantized prior to processing. The precision used to represent each of the picture elements of the image, or each of the filter coefficients, determines the truncation error that is introduced.

Simulation results show the capability to perform the transform at video rate. The effect of finite precision processing does not significantly affect the quality of the reconstructed images. Figure 27 shows an original image Lena 256×256 pixels, the reconstruction with a finite precision of 8 bits (on ASP), a compressed image with floating-point precision, and a compressed image with a finite precision of 8 bits (on ASP). The corresponding computation time is 65 ms on the ASP with 64K processors. Simulations pointed out the importance of I/O, and the need to parallelize this task.

Figure 27 Simulation results on the test image Lena: (a) original image 256 × 256 pixels, 8-bit value per pixel; (b) reconstructed with finite precision of 8 bits, PSNR = 39.3 dB; (c) compressed 8 : 1, floating points, PSNR = 32.26 dB; (d) compressed 8 : 1, 8-bit precision, PSNR = 29.02 dB.

7.1.4.2 Neural autoassociation for image compression: a massively parallel implementation

Image compression via linear and nonlinear neural networks turns out to be effective in terms of high compression ratios and reduced image quality degradation, despite of the simple architecture employed. Generally, a three-layer perceptron in autoassociative mode (the teacher vector is identical to the input vector, thus the net learns the identical mapping) is used (Fig. 28). An image is sampled in $k \times k$ blocks to form a vector of k^2 elements by row-wise raster-scanning. The network performs data compaction of the input since there are fewer neurons in the hidden layer than in the input or output layer. The number of neurons in the input and in the output layer is identical.

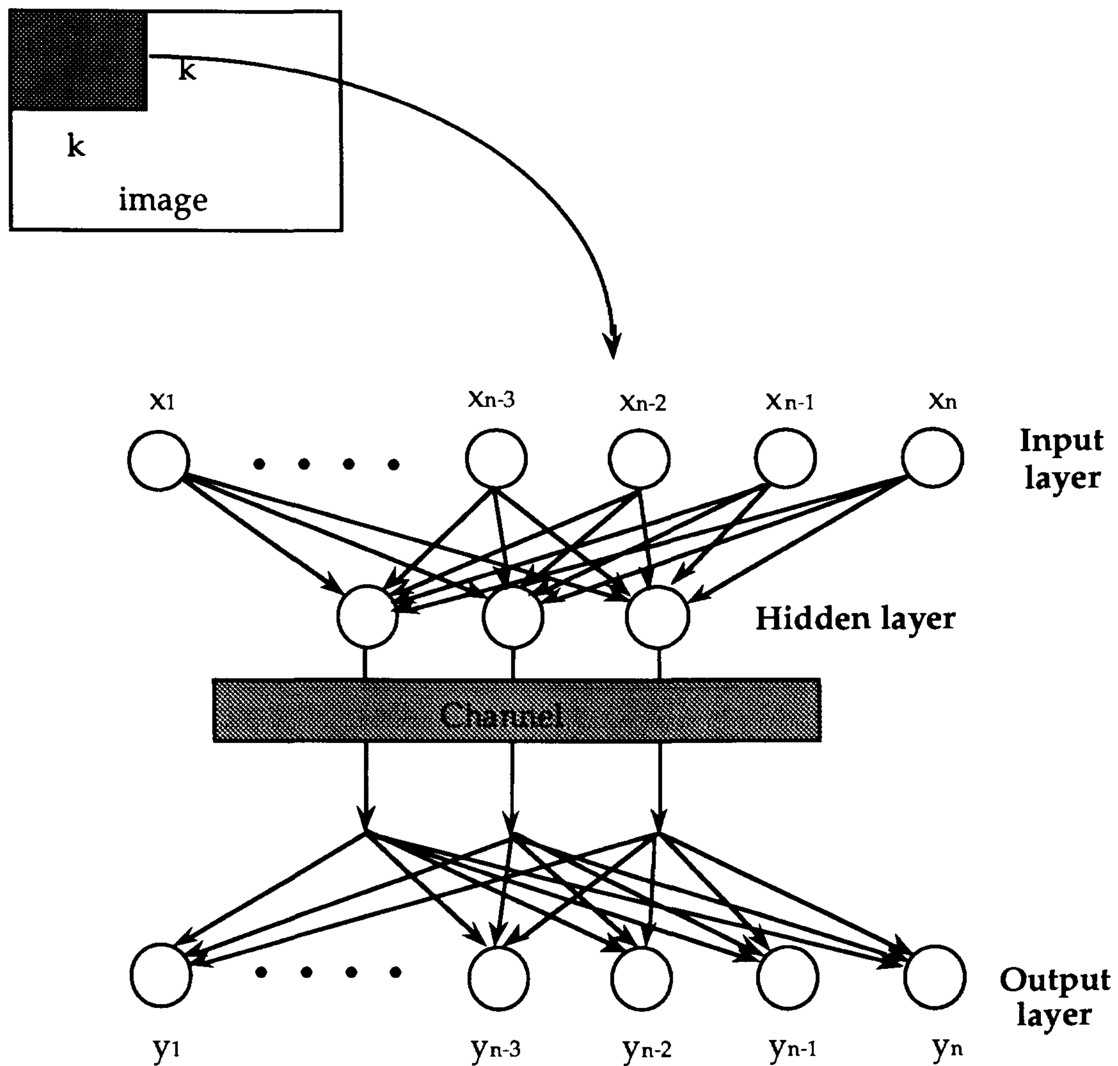


Figure 28 The three-layer perceptron for image compression

In the learning phase, which is unsupervised due to the autoassociative mode, the network is forced to compute a set of good, hidden-layer weighting factors to represent the input data. The output layer uses this internal representation for the reconstruction of the input pattern. The transmission channel can be assumed to be present immediately after the hidden-layer neurons.

After the work of Cottrell et al. [40] which opened the exploration of a possible utilization of multilayer perceptrons for image compression, many efforts [41]–[43] were made to clarify the behaviour of the network and the role of the different parameters involved such as the number of neurons in the hidden layer used to evaluate the error [44], and the use of other norms [45]. In summary, the basic network shown in Fig. 28 performs, in the linear case, a transformation similar to the Karhunen–Loève Transform (KLT). The results given by the neural network are comparable with standard algorithms but with the advantage of having a simple implementation. Further improvements [41] in

terms of compression ratio and signal-over-noise ratio (SNR) have been obtained using a hierarchical framework, where a classification of the input vectors is performed a priori and each vector is processed with specialized subnetworks.

The neural model shown in Fig. 28 has been mapped in the linear structure of the ASP and activity bits are used as delimiters for layers and neurons. A three-layer ($64 \times 8 \times 64$) perceptron has been employed and fixed-point arithmetic has been used. The structure of a single neuron is sketched in Fig. 29.

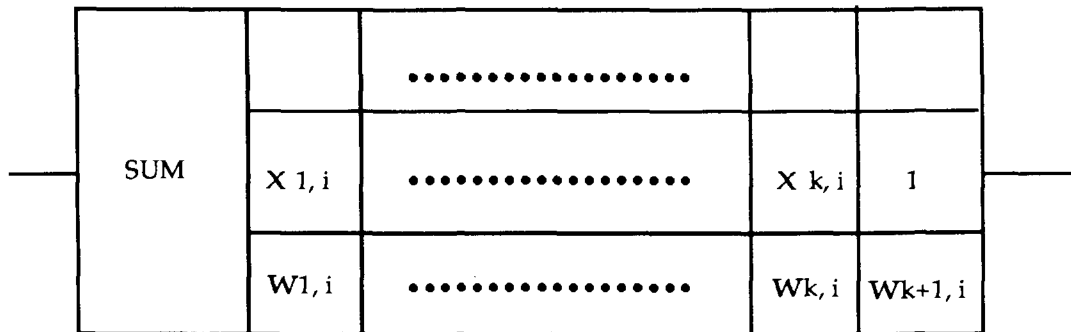


Figure 29 Data-register distribution of component APEs of a neuron

Each neuron is composed of a number of APEs equal to the number of neurons of the previous layer, plus one APE (SUM) for storage of the neuron output, and another for the bias. The generic layer i, composed of l_i neurons, requires $l_i \times l_{i+1} + 2$ APEs. Globally, the number of APEs needed is $\Sigma_i l_i.(l_{i+1} + 2)$where the sum is done over all the layers of the network. Each 64-bit data register is partitioned into three fields. The first field, 16 bits long, is devoted to the storage of weight coefficients of the neuron. A 16-bit precision for weight representation is shown to be sufficient for convergence in most classical examples of back propagation of the input value (hidden layer), or for output of the previous layer's neurons (output layer). The third field is used for temporary storage.

Back propagation consists of two steps. During the feed-forward pass the input data are propagated forward through the synoptical connections from the input layer to the output layer. During the backward pass the error calculated at the output layer is propagated backwards.

Each layer is processed in parallel, one layer at a time. After a parallel load of each component of the input vector, a parallel multiplication occurs between inputs and weights, and the result is accumulated in the third field. By means of a hierarchical addition the total weighted sum is stored in the SUM APE of each neuron. The result from the SUM APE is then broadcast to the subsequent layer. The process is repeated for all layers. For the feedback pass, the error at the output layer is calculated and weight correction is performed. After a broadcasting of the error to the hidden layer, the weights are properly changed in order to decrease the error vector.

The simulation results obtained are shown in Fig. 30. A 256×256, 8-bit, gray-scale test image sampled in 8×8 non-overlapping blocks has been compressed and reconstructed. The compression ratio is 8 and the PSNR is 34.7 dB. The learning required 200 epochs (a training-set dimension of 2048 vectors coming from four different images). The learning speed is of the order of 10^7 connections per second.

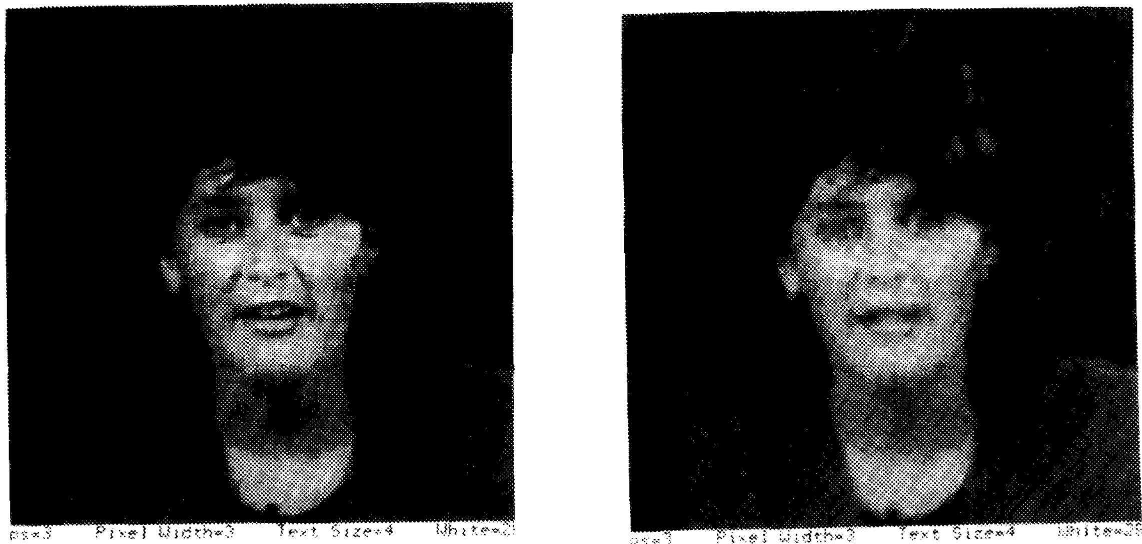


Figure 30 Original (left) and reconstructed (right) image. Compression ratio: 8.

The VASP simulator was used for studying this implementation. It was configured with 64K APEs. Each $64 \times 8 \times 64$ network requires 714 APEs. Ninety-one nets were implemented and worked in parallel. Each net was able to compute 500 64-element vectors per second. Globally, 45 500 vectors per second were processed. Thus real-time compression and decompression is possible.

7.1.4.3 Conclusions concerning the use of ASP for image compression

Two methods have been described, the first one based on a Gabor-like wavelet transform and the second one on a neural-network-based technique. Massively parallel implementations on the ASP have been presented. Results in terms of computational speed and image quality reconstruction were discussed: they show the feasibility of real-time video compression using ASP architecture.

7.2 Application studies on ASTRA machines

In this section, we report on the development of algorithms which were developed on the simulator and later installed on the ASTRA machine. The timing results have been obtained from the work performed on the real ASTRA machine running with different kinds of operating systems as explained later.

7.2.1 Online data-processing in a high-energy physics experiment

A real-time application for the ASPs has been completely designed, implemented and run at LAL-Orsay [46]. We used a standard LAC and an ASPA board (2K processors), controlled by a FIC8232 processor [11] running the OS-9 operating system [47]. The aim of this work was to evaluate the computing power and flexibility of the ASPs in an existing online environment.

We used the data from the RD3 experiment at CERN. This experiment is testing a prototype calorimeter for a future experiment at LHC. The data acquisition chain of RD3 has been simulated with a Sun workstation: real data, stored on disk, were sent in blocks of 2 Mbytes to the FIC via an SVIC/VIC interconnect system [9]. A first task of the FIC was to collect the data. A second task was the decoding of these data to send them, event by event, to the input buffer of the ASP board; it also pushed a request for processing by the ASPs into the relevant fifo of the LAC. A third task was to collect results from the output buffer of the ASP board, and a fourth task sent back to the Sun the results corresponding to the 2 Mbytes block input. The four tasks were running asynchronously, and each of them continuously estimated the time spent working, waiting for a source, or waiting for a destination. Because of the limited input rate via VME, we observed that the ASPs were idle more than 90% of the time.

The ASP processing involved a special loading procedure whereby the address of the processor to load the next data was read from the processor receiving the previous data. Then different procedures involved parallel integer additions, multiplications, maximum finding, and a vector-scalar division, with results in fixed-point format. We could not run the VASP chips with a time slot less than 100 ns (we were limited in speed by the ES2 chip which ran four times more slowly than specified). As a result, the time needed for a multiply and accumulate operation on 16-bit integers was measured to be ~ 270 µs. Finding a maximum is rather fast (26 µs in this application). However, to prevent two different PEs from being marked simultaneously as maximum involves a search along the string which takes 350 µs. Once this search is done, relevant data may be read only from the PE concerned. This takes less than 10 µs.

From this interesting experiment, we concluded that, apart from a large number of minor bugs in the lamc (being corrected now), the application developed at LAL-Orsay could be run reliably. However, to be really competitive with other techniques, a higher

integration in the PEs themselves, in the I/O channels, and in the low-level control is necessary.

7.2.2 The SDC second-level trigger

Following the first studies made on a simulator, a new architecture has been designed for the second-level trigger for the SDC detector [48] at the Superconducting Supercollider. This architecture contains several steps, the first one consisting in an extraction of objects from the large amount of data coming from the detectors: this task must be done in a full η (pseudo-rapidity)/ϕ (azimuth angle) mapping in some tens of microseconds only.

An event collector will use the lists of physical objects coming from each algorithm in order to define cells of interest. A global decision will be taken at the next step by comparing the pattern with a trigger reference-list.

Algorithms of object extraction have been programmed on an ASTRA machine; the ASTRA machine is used to run different algorithms on data from simulated events generated by a Monte Carlo simulation tuned for the SDC detector, in order to choose and optimize the most efficient algorithm for the trigger.

One implementation of these algorithms uses the Shower-max data and the tracking information: the goal is to find peaks of energy in each segment of 1024 pixels in ϕ. The calculation is done in parallel in the 30 segments defined in η. For each peak found, the program performs the energy summation and separates photons and electrons by looking at the tracking information in the corresponding pixel, i.e. in each APE. The output is a list of objects to be given to the next step of the triggering system. This program, containing thresholding, maximum finding, energy summation and matching with tracks, runs in 10 μs to process one event. Owing to the fine-grain SIMD associative architecture, a feature of the ASP, this time does not depend upon the number of data channels.

Another algorithm for object extraction is a cluster-finding using either the electromagnetic calorimeter data, or the hadronic calorimeter data (each one with its own granularity); the goal of this algorithm is to find a localization of electron or hadron clusters, to calculate the sum of energy for each cluster, and, to be able to count the clusters, to reduce each cluster to one pixel correctly located. The way to do this pixel cluster reduction is to compare the local value of energy with the eight neighbours and to keep the pixel if it is the maximum. This program runs in 15 μs for processing an event, independently of the number of data.

A lot of simulated events are under study on ASTRA in order to accumulate enough statistics to evaluate the different algorithms on various types of events. This global work is expected to help us for the proposal on the second-level trigger architecture of SDC, and the complete simulation of the trigger is under development.

The ASPEN machine is also used to evaluate the performance of the second-level trigger algorithms of the SDC experiment.

7.2.3 Image processing for peak-finding from cluster data

In many physics experiments the huge amount of information collected from interactions is stored in a very efficient way in images recorded by film or CCD cameras. In this range of applications, at Saclay one peculiar ASP application has been studied and evaluated for an astrophysics experiment MACHOS [49], [50]. It involves the analysis of very large images of the stars lying in the Magellanic Cloud (1.2 Gpixels, 12 bits per pixel). As another example, ASPs have also been evaluated for tracking particles in a heavy-ion experiment (CERN-NA35) using stereo streamer chamber pictures [51], [52]. Finally, Geneva University was also interested to use ASPs for image processing in another heavy-ion physics experiment (CERN-WA93).

As a typical image-processing ASP application running on the ASTRA machine, we will now recount the main results of the work done at CERN for the WA93 experiment.

The tracking of the charged particles in the WA93 experiment is done by using a set of luminous chambers which sample the particle trajectories at several planes. CCD cameras, looking at the luminous planes, directly record the space-point spots left by the particles crossing the chamber planes for the subsequent tracking analysis. Here we would like to concentrate on the first step of the tracking analysis, the so-called Peak-Finding (PF). This name originates from the fact that the particle hits are generally random cluster patterns on the CCD-pixel planes. The position of the particle is identified by the pixel with the highest light yield.

7.2.3.1 The peak-finding algorithm

The peak-finding algorithm is an image processing algorithm for image feature extraction. In particular, it filters an image, representing clusters generated by secondary particles emerging from heavy-ion interactions, into a simplified image containing single pixels corresponding to the peaks inside the clusters.

The PF algorithm is divided into two consecutive parts: the weighting and the peak selection procedures.

– The weighting procedure

The weighting procedure is used to identify those pixels of the image that belong to any possible track hit cluster.

The following steps are performed (see Fig. 31).

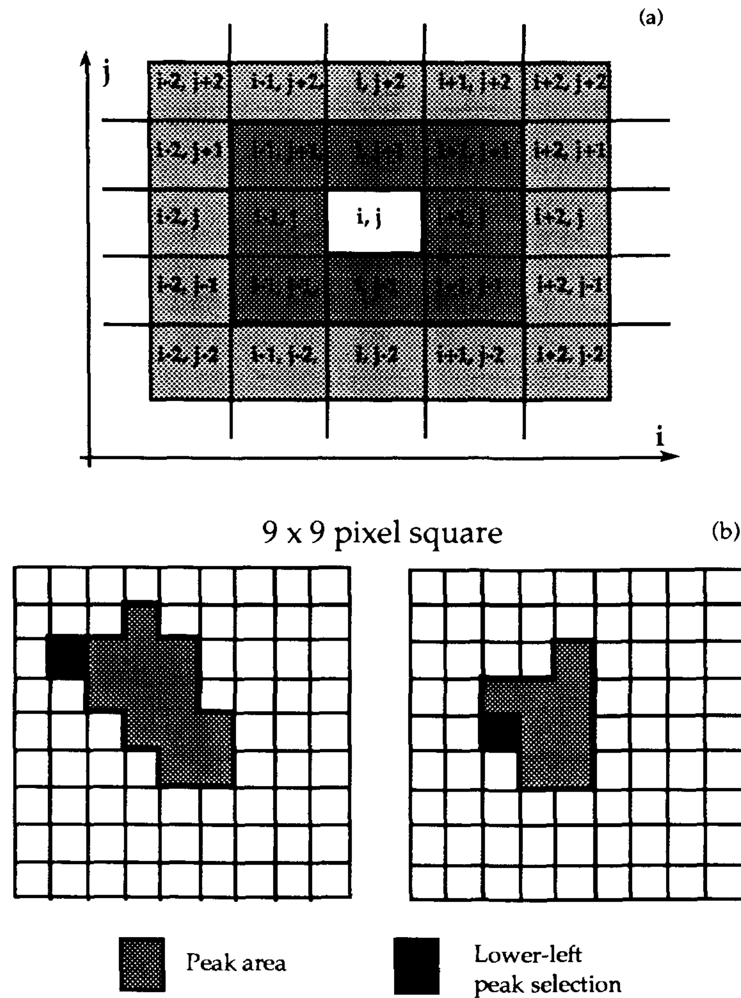


Figure 31 (a) Pattern analysis around each pixel, (b) lower-left pixel definition for peak finding

(1) In parallel, for each pixel, the amplitudes are summarized on a ring-by-ring basis (i, j runs over the full image size):

$$w_1(i,j) = a(i,j+1) + a(i,j-1) + a(i-1,j+1) + a(i-1,j) + a(i-1,j-1) + a(i+1,J+1) + a(i+1,j) + a(i+1,j-1)$$

$$w_2(i,j) = a(i,j+2) + a(i,j-2) + a(i-2,j+1) + a(i-2,j) + a(i-2,j-1) + a(i-2,j+2) + a(i-2,j-2) + a(i+2,j+1) + a(i+2,j) + a(i+2,j-1) + a(i+2,j+2) + a(i+2,j-2),$$

where $a(i,j)$ is the amplitude of the pixel of coordinates i and j.

(2) In parallel, for each pixel of the image, count the number of pixels with non-zero value in the two surrounding rings separately around each pixel:

$$c_1(i,j) = b(i,j+1) + b(i,j-1) + b(i-1,j+1) + b(i-1,j) + b(i-1,j-1) + b(i+1,J+1) + b(i+1,j) + b(i+1,j-1)$$

$$c_2(i,j) = b(i,j+2) + b(i,j-2) + b(i-2,j+1) + b(i-2,j) + b(i-2,j-1) + b(i-2,j+2) + b(i-2,j-2) + b(i+2,j+1) + b(i+2,j) + b(i+2,j-1) + b(i+2,j+2) + b(i+2,j-2),$$

where $b(i,j) = 1$ if $a(i,j) > 0$, 0 otherwise.
(3) The following criteria are applied:
Pixels above a predefined threshold, which have enough neighbours counted in the two rings with a total amplitude above a predefined threshold on a ring-by-ring basis are kept and are called activated pixels:

$$a(i,j) > t_0$$
$$w_1 > t_1 \quad \text{and} \quad w_2 > t_2$$
$$c_1 > n_1 \quad \text{and} \quad c_2 > n_2$$

The t_0, t_1, t_2, n_1, and n_2 values are determined empirically.

– *The peak selection procedure*

In parallel, find the maximum value among all those activated pixels inside a 9×9 neighbourhood square. Those pixels selected with this procedure are called local maxima. In the case where more than one maximum is found, only the lower-left pixel inside the maxima area is kept.

Mark as peaks all those local maxima which have at least one activated neighbour.

7.2.3.2 Algorithm timing results

The algorithm was previously studied and implemented for the simulator at the University of Geneva [53]. The algorithm was then implemented by CERN and tested on the CERN ASTRA machine. The measured processing time for an image of 32×64 pixels was 2.3 ms for the machine running at 10 MHz clock rate. When considering the input loading time, the processing time, and the output overhead, the overall duration was close to 10 ms.

The processing time measured in simulation, assuming a 25 ns time slot for micro-instructions, was close to 1 ms for an image of 256×384 pixels (98304 APEs string). For a larger picture, assuming a bigger string size, the processing time does not change significantly. This is not the case for the I/O operations which increase proportionally to the number of pixels to be loaded and dumped. For a very large number of data, the comparison between processing time and I/O overhead shows the need to realize a machine interface architecture which supports parallel I/O operations.

7.3 ASPEN evaluation in the NA48 experiment

A trigger system based on the ASPEN architecture is under design for the level-2 charged trigger of the NA48 CERN-SPS experiment (expected to run in 1995). The task is to associate the 4 XYUV wire planes to get two-dimensional space points. The process finds points which are measured in all the projections; it also accepts incomplete three-plane combinations which result from detection inefficiencies in the chamber.

The fully sequential version of this matching algorithm requires 'compute and match' association loops to compare the reconstruction from two projections with the actual measurements from the other two projections. This computation is achieved in several steps to ensure that all potential points with only three measurements are found. This algorithm uses multiple nested loops resulting in n^3 dependency for the response time where n is the track multiplicity.

A parallel version of the XYUV algorithms is currently under test on the ASPEN machine, used as a geometric look-up table. The ASP array is a geometric image of the detector. The chamber is binned in X, Y, U, and V intervals. Each resulting region is allocated to a single APE which is preloaded with is own XYUV position. For each event, the X, Y, U, and V values are successively proposed to the array; each APE stores the match information for each projection and the index. The array is then read back to obtain the indexes from the matching APE. This algorithm can flag at once the 3-coordinate and the 4-coordinate points. This process has a dependency in n on track multiplicity. Several ASPEN machines will be used to define the drift chamber space-points from the corresponding projective measurements.

8. PERSPECTIVES

The main constraint for level-2 trigger processing is time. The overall processing time spent in a system may be split into three parts:

- Loading the data to be processed
- Processing the data
- Reading the results

In all massively parallel systems, the time to load a large amount of data becomes a major part of the process when the time allowed to process data is relatively short. In the case of sparse data, the associative property of the ASP allows the loading of each datum directly into the right APE. The process is first to activate the corresponding pixel(s) then write data in the selected APE(s) (list loading). In case of a large file of contiguous data (as for a calorimeter), in which you cannot separate background noise from particles at minimum ionization like muons, data are to be fed sequentially into each APE, bringing some dead time.

8.1 A new chip: the VASP-128

A new chip, see Fig. 32, is under development to avoid this dead-time problem: an additional vector data buffer can be loaded during the processing of the previous event. Data can simultaneously be transferred in parallel from this buffer to all APEs. The first

batch of this chip is now under test at ASPEX and most of the new features are working properly.

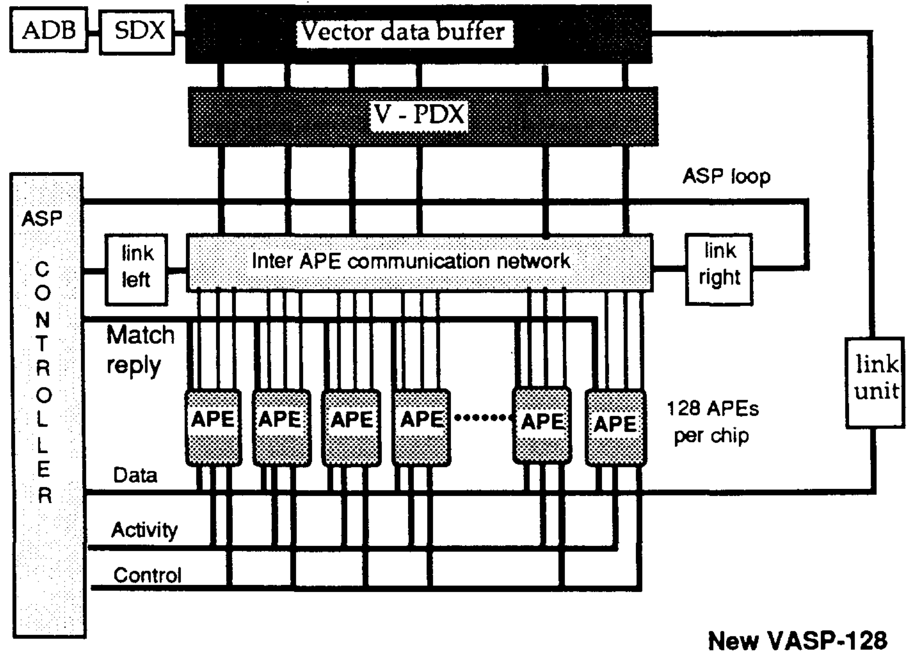


Figure 32 The new VASP-128 chip with a vector data buffer

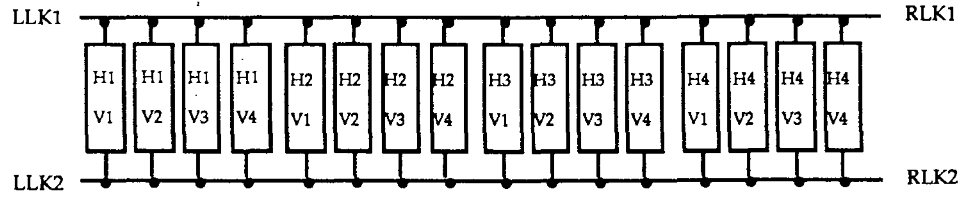
The VASP-128 offers other new features such as the implementation of a second inter-APE network (on Tr2 flag), a bypass every 16/64 APE, a 16-bit data format, a shifting by 1 or by 4, and others.

8.2 Development of a two-dimensional ASP

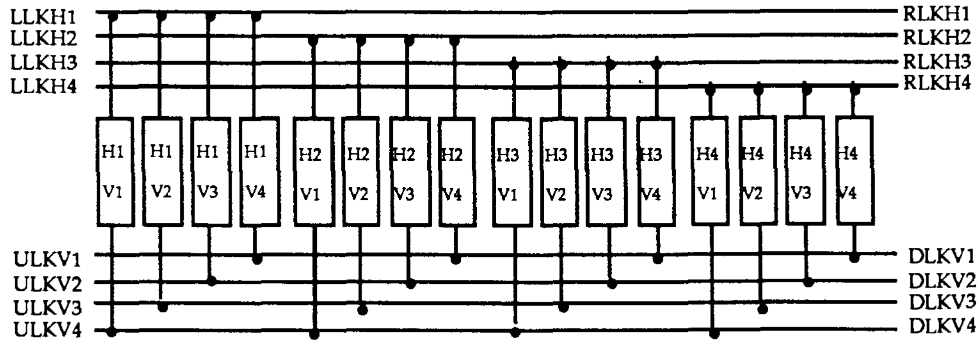
An event in a LHC/SSC detector is a three-dimensional image. By writing all the projective data in the same processor this image becomes a 2D image. It is straightforward to map a 2D image in a processor string by splitting the string into segments.

The communications network of the ASP allows to transfer data in both the horizontal and the vertical direction by using the shifting operation along the string. If the detector mapping is done line by line along x, the communication to horizontal neighbours is easy and fast (shift only by one APE), but the vertical communication is time-consuming in asking for a large number of shifts (one full line each time). The R&D plan is to modify the new VASP-128 chip, taking into account the two communications networks existing in the chip to wire one network in the horizontal direction and the other to form a vertical path (Fig. 33).

Current chip: Linear presentation



Modified chip: Linear presentation



Modified chip: x - y presentation

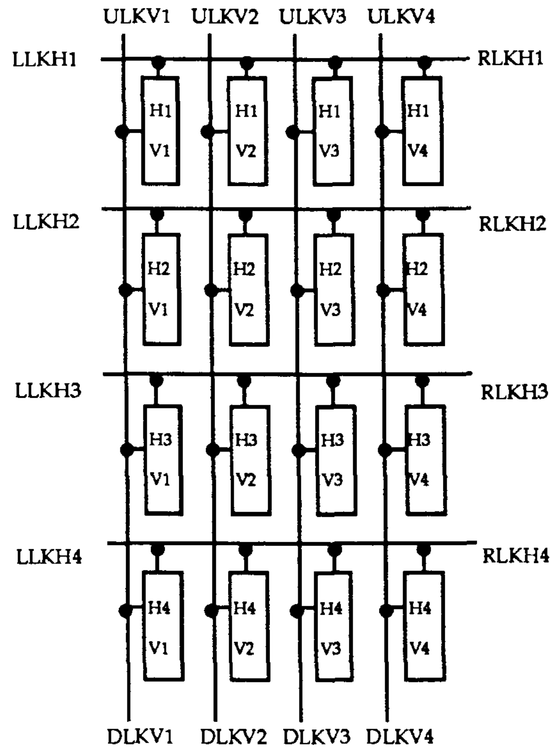


Figure 33 Example of modification to get a two-dimensional APE

9. SUMMARY AND CONCLUSIONS

A number of goals defined in the original project have been achieved.

- The ASTRA machine is operational with a full operating system, and has been used for assessing the potential of the ASP on several applications.
- ASPEN, as a dedicated real-time machine, was designed, built, and used for the evaluation of a trigger in a physics experiment.
- A high-density 8K-APE board, the HASPA board, equipped with four parallel 32-bit input ports, has been designed and built.
- VASP chips, working at a 25 ns time slot, are now readily available.
- An upgraded chip prototype with a vector data buffer and a higher processing element density has been produced and is at present under test at ASPEX.
- A friendly software environment, well documented, is working and supported, making ASP machines easy for algorithm development for actual applications.

The ASP concept for application programming was easily understood. This original concept was used by physicists and engineers and found efficient in many domains. The necessary know-how for low-level ASP programming was acquired by the collaboration in a relatively short time.

For real-time processing in physics experiments, we measured the performance of representative elementary algorithms relevant for level-2 trigger decisions: execution times were found to lie in the range between 10 μs and 50 μs. We have verified that ASPs are much better suited to the resolution of topological problems involved in those fast algorithms than for numerical calculations.

A SIMD-based trigger architecture could take advantage of a basic multiprocessor chip, surrounding it with custom-made hardware, and using multiple SIMD subsystems in a pipeline. Such SIMD systems, however, are not on the market at present, and development efforts are still necessary. Some comparisons with other systems are being performed in another R&D project [34].

When trying to reach a very high density of APEs per board, we encountered two difficult problems:

- fabrication of large dense and fast hybrid modules;
- implementation of a sufficient number of I/O ports adapted to the large processing power of the board.

However, with the HASPA board, its four parallel input ports, and the use of double-page fast buffers, we demonstrated that satisfactory compromises may be made between the processor density and the I/O capabilities for a wide range of applications.

The main conclusion of our project is that the ASP architecture is well adapted for real-time algorithms of a morphological nature. Thus, it can contribute to solving physics-feature extraction problems in domains where local operations and associative searches are dominant.

In fast triggering for future hadron colliders, the programmability of SIMD processors like ASPs may turn out to be an essential requirement. This makes them flexible components which can be adapted to different kinds of elementary algorithms. At this point, we should underline that 'programming' in this context is not simply programming in a high-level language for a general-purpose computer. It requires a new way of thinking for the design of algorithms that exploit both the parallelism inherent in the problem and that available on the processing machine; this is also true for any processor expected to run algorithms at rates above 10 kHz. In the case of the ASPs, it implies the knowledge of the computing architecture and the learning of a low-level language which consists of a set of built-in procedures to be used within a high-level programming language.

The efficiency of ASP for image-compression techniques has been shown. Results in terms of computational speed and image-quality reconstruction show the feasibility of real-time video compression using an ASP machine.

For the future, an ASTRA-User physics community was created at CERN after the end of the MPPC project. ASPEX and CEA/CEN Saclay are pursuing ASP R&D:

- dedicated ASP machines, tuned to physics experiments are being designed;
- future chips, endowed with new capabilities and improved performance, are under development and will make this architecture even more attractive in the years to come. We would like to underline the expected improvement of the chip speed by the progress in technology and the improvement of the algorithm efficiency by the architecture upgrade.

Acknowledgements

We thank all the people from the collaborating institutes having worked for this project who are not formally listed in the MPPC Collaboration.

We want to acknowledge at CERN, P. Darriulat, Director of Research; H. Wenninger, Division leader of the former EF division; C. Fabjan and A. Sandoval for their particular help and support during the preparation of this project in 1988 and 1989. We would also like to thank particularly Professor P. Lehmann, past Director of IN2P3 in France, and Professor B. Vittoz, President of the Ecole Polytechnique Fédérale de Lausanne in Switzerland, for their essential support when launching this project.

At CERN, our students G. Bressani and C. D. Moffat must be acknowledged for their work and enthusiasm during their participation in this project. We also thank P. G. Innocenti, CERN-ECP Division leader, for his interest and support.

The DAPNIA-Saclay group thanks Ph. Briet, P. Peyraud, M. Barats, M. Milisic, and P. Bouyer for their contribution in the implementation and fabrication of the ASP circuit boards, and A. Hauviller for the organization of many meetings.

The LAL-Orsay group wants to thank C. Eder, J. P. Coulon, J. Daubin, and C. Caresche for their contributions to the project.

Our Hungarian colleagues would like to mention that their contribution was partially supported by the Hungarian grants OKTA-3271 and OKTA-4092.

APPENDIX

The goal of the following programming examples is to provide the interested reader with a good basis for understanding programming principles, and an introduction to the programming style. The reader should have basic experience of the Modula-2 programming language and of the UNIX environment.

A.1 FIRST EXAMPLE: WRITING THE SKELETON OF AN ASTRA PROGRAM

Let us write an ASTRA program called 'tutor' (like a 'Hello World!') which makes a call to the low-level ASP controller and return. According to the idea introduced in Section 5.1 we write the ASTRA application starting from the lowest level modules (LAC modules). The following steps must be performed.

Edit a definition and an implementation module for the LAC level as follows:

```
%textedit  tutorlac.def
```

```
(*$$ Low Level ASP*)DEFINITION MODULE tutorlac;
(*$$ EXPORT CROSS LEVEL
              lacproc; *)
PROCEDURE lacproc ();
END tutorlac.
```

Notice that the comments marked with `$$` are special instructions for the aspc compiler. In this particular case the statement `(*$$ Low Level ASP*)` tells the aspc compiler about the target hardware level (i.e. LAC). The statement `(*$$ EXPORT CROSS LEVEL lacproc; *)` tells the aspc compiler which symbol must be cross-exported to the higher level (i.e. IAC). In this way the procedure `lacproc()` can be imported and invoked in the IAC-level code.

```
%textedit  tutorlac.mod
```

```
(*$$ Low Level ASP*)IMPLEMENTATION MODULE tutorlac;
CONST apesInString = 2048;
PROCEDURE lacproc ();
```

```
BEGIN
   TargetAsp("ASTRA_E1_124");
   AssumeWorstCaseSegmentLength(apesInString);
   AssumeInterChipDelay(25);
   ConfigureString(Chain);
   Declaration(sf{12..17}, sf{18..20}, sf{22..32});
   Reset(Inc, ro{p1,p2,p3, m1,m2,m3});
END lacproc ;
END tutorlac.
```

The *lacproc()* performs some system instructions then formats the data registers to three different serial fields, resets the markers, the pointers and returns. For more details about these LAC primitives see Refs. [16] and [17].

Then compile these modules as follows:

```
%aspc  tutorlac.def
%aspc  tutorlac.mod
```

This command will generate the target code for the LAC level and the cross-level interface code for the IAC level of *lacproc()*.

Edit a definition and an implementation module for the IAC level as follows:

```
%textedit  tutoriac.def
```

```
(*$$ Intermediate Level ASP*)DEFINITION MODULE tutoriac;
(*$$ EXPORT CROSS LEVEL
     iacproc; *)
PROCEDURE iacproc();
END tutoriac.
```

As noted above for the LAC modules, the aspc macro instructions *(*$$ Intermediate Level ASP*)* allow to compile for the right target machine level and to generate the upper cross-level interface (in this case for the HAC level).

```
%textedit  tutoriac.mod
```

```
(*$$ Intermediate Level ASP*)IMPLEMENTATION MODULE tutoriac;
FROM tutorlac
   IMPORT lacproc;
FROM xpcAndDmaStatus
   IMPORT tPid, Schedule, Wait;
PROCEDURE iacproc()
VAR pid: tPid;
BEGIN
   (* Remote Call to LAC *)
   Schedule(lacproc(), pid);
   (* Wait lacproc() to return *)
   Wait (pid);
END iacproc;
END tutoriac.
```

The `lacproc()` is imported with a normal Modula-2 statement from the lower module tutorlac. Notice that only those LAC symbols which are part of the `EXPORT CROSS LEVEL` list can be imported into the IAC implementation modules.

Since the procedures imported from a LAC module are not a normal procedure call but a cross (remote) procedure call to the LAC level, they must be invoked with some special instructions.

The `xpcAndDmaStatus` library module provides a set of routines for Cross Procedure Call management. The `Schedule` routine calls a Cross Procedure Call and returns the `pid` (procedure identifier) for that call. The `pid` is used for the procedure return synchronization. The `Wait` procedure blocks the execution until the `pid` corresponding to a Cross Procedure Call is returned. That is, in this particular case, whether the `lacproc()` is terminated or not.

Compile these modules as follows:

```
%aspc  tutoriac.def
%aspc  tutoriac.mod
```

This command will generate the target code for the IAC level and the cross-level interface code for the HAC level of `iacproc()`.

Edit a main module for the HAC level as follows:

```
%textedit  tutorhac.mod
```

```
(*$$ High Level ASP*)MODULE tutorhac;
FROM tutoriac
   IMPORT iacproc;
FROM xpcStatus
   IMPORT tPid, Schedule, Wait;
VAR pid: tPid;

BEGIN
   (*Call the Remote iacproc() *)
   Schedule(iacproc(), pid);
   (*Wait iacproc to return *)
   Wait(pid);
END tutorhac.
```

Notice that the way to use the Cross Procedure Call is the same as that shown for the IAC to LAC level, except that the `Schedule` and the `Wait` routines are imported from a module called `xpcStatus`. In the next example the difference between the two modules will be explained.

Compile this module as follows:

```
%aspc  tutorhac.mod
```

Then link the application with the following command:

```
%aspl  -o  tutorhac  tutorhac
```

The command will generate the three-level executable module.

Execute the tutor program by typing:

```
%tutorhac
```

The tutor program will execute according to the following scheme:

HAC		IAC		LAC
Main				
Schedule of iacproc	⇒	iacproc		
		Schedule of lacproc	⇒	lacproc
				lac primitives
			⇐	Return
		Wait lacproc		
Wait iacproc	⇐	Return		
Exit tutor program				

Before the effective execution of the user code, the system checks the status of the ASTRA machine. In particular, it initializes the different hardware components and downloads the necessary code to the IAC and LAC levels. The way the target code is downloaded to the lowest level IACs and LACs and is executed is a task of the ASTRA operating system tools.

The tutorial example above can be considered as the simplest ASTRA application. It just performs the minimum set of instructions which apply to all levels: HAC, IAC, and LAC. This example gives an idea of how the Cross Procedure Call mechanism is realized. The next section will give a more complex example which implements a real algorithm on the ASP and on the data passing between the three levels.

A.2 SECOND EXAMPLE: AN EXTENDED PROGRAM, *ARRAY OF SUMS*

We want to implement the algorithm *Array of Sums*, as explained in Ref. [24], for the three-level machine.

The idea is to create an interface procedure called *pSums* (Parallel Sum) with a parameter indicating the pointer of the array in which we want to calculate the sums. The picture below shows the Modula-2 declaration of *pSums*.

Given the array:

$P1$	$P2$	....	Pn
$a1$	$a2$		an

the *pSums* procedure returns

$P1$	$P2$	....	Pn
$a1$	$a1+a2$		$a1+a2+...+an$

where n is the number of elements (in this example we assume $n = 64$).

According to the algorithm, the data vector *a1...an* must be loaded into the Associative String for the computation and then dumped for the result. The system so defined needs data passing between the three levels. Let us introduce the tools which support the programmer in this task.

A.2.1 Passing data between HAC and IAC

In order to pass the data array to the IAC level and get the result, we make use of a shared data area between HAC and IAC.

The programmer can allocate and de-allocate several pieces of memory by using a set of interface routines supported by the ASTRA operating system tools. The interface routines can be accessed by importing the *SharedStorage* library module in our application (see Ref. [18]).

The area of memory allocated in the HAC-level module becomes a real shared area when the accessing address (pointer) is passed to the IAC level. In particular, the pointer is passed to the IAC as a special value parameter of a Cross Procedure Call.

The programmer has to declare a parameter of a predefined type called *tSharedMemAdd*. The pointer to the shared data in the HAC address space is translated into the pointer in the IAC address space during the Cross Procedure Call parameter passing.

A.2.2 Passing data between IAC and LAC

The LAC hardware is equipped with a set of I/O fifo Scalar Data Buffers (SDBs) for loading and dumping data between LAC and IAC. The basic idea is to load the data array into the LAC input fifos and pass the number of items to the LAC as a Cross Procedure Call special parameter. A similar approach is used for reading the array.

A set of library routines support the programmer for accessing these fifo with Direct Memory Access (DMA) transfers. These routines can be imported from the library module *sdbHandler* (see Ref. [18]).

Other methods can be used for data I/O between the IAC and the LAC. For more details see Ref. [18].

A.2.3 The pSums implementation

Let us write the *pSums* system starting from the LAC until the HAC module.

% textedit psumlac.def

```
(*$$ LowLevel ASP *) DEFINITION MODULE psumlac;
FROM lamTypes
IMPORT CtrlParameter;
(*$$ EXPORT CROSS LEVEL
   InitApeArray,
   LoadApeArray,
   DumpApeArray,
   ComputeSums; *)
PROCEDURE InitApeArray();
PROCEDURE LoadApeArray(Count: CtrlParameter);
PROCEDURE DumpApeArray(Count: CtrlParameter);
PROCEDURE ComputeSums();
END psumlac.
```

Notice that four procedures are cross-exported to the IAC level for initialization, I/O, and computing. The parameter `Count` of type `CtrlParameter`, used in the load and dump routines, is a special type (CARDINAL) which indicates the number of elements of the array. It is used to count the number of pushes and pops to be done from the LAC SDB fifos.

% textedit psumlac.mod

```
(*$$ LowLevel ASP *) IMPLEMENTATION MODULE psumlac;
CONST AspSize = 2048;
PROCEDURE InitApeArray();
BEGIN
   TargetAsp("ASTRA_E1_124");
   AssumeWorstCaseSegmentLength(AspSize);
   AssumeInterChipDelay(25);
```

```
  ConfigureString(Chain);
  WriteSegLinks (s64);
END InitApeArray;
PROCEDURE LoadApeArray(Count: CtrlParameter);
BEGIN
  SetLeftLink(TRUE);
  Uncond(TagBit(bm{}, sd{}, ab{}, trl));
  WordWrite(NoClr, slClosed, a , bm{}, wf0, 0, ab{a10,a20,a30,a40});
  WordWrite(NoClr, slClosed, a , bm{}, wf1, 0, ab{a50, a60});
  Uncond(TagBit(bm{}, sd{}, ab{a11}, trl));
  TagShift(slClosed, 1);
  SetLeftLink(FALSE);
  SetRightLink(FALSE);
(*
* Load ASP substring
*)
  FOR 1 TO Count DO
    WordWrite(NoClr, slOpen, a, bm{}, wf0, swFifo, ab{});
    TagShift(slClosed, 1);
  END;
END LoadApeArray;
```

The `LoadApeaArray` procedure loads each element of the Associative String with the data coming from the LAC input fifo `swFifo`. In particular, each item from the `swFifo` is placed into the first word (4 bytes) of the data register of each APE.

psumlac.mod continues ...

```
PROCEDURE DumpApeArray(Count: CtrlParameter);
BEGIN
  SetLeftLink(TRUE);
  Uncond(TagBit(bm{}, sd{}, ab{}, trl));
  BitWrite(NoClr, slOpen, a, bm{}, sd{}, ab{a10});
  Uncond(TagBit(bm{}, sd{}, ab{a11}, trl));
  TagShift(slClosed, 1);
  SetLeftLink(FALSE);
```

```
(*
* Dump ASP string
*)
  FOR 1 TO Count DO
    WordRead(NoClr, slOpen, a, wf0, dcr, bm{}, ab{});
    AssignDcrTo(rdFifo);
    TagShift(slClosed, 1);
  END;
END DumpApeArray;
```

The procedure `DumpApeaArray` reads the first word of each APE of the string and pushes it into the LAC output fifo `rdFifo`. Notice that the word coming from the APE, before the loading into the `rdFifo`, must be converted through the data conversion register (*Dcr*).

The procedure `ComputeSums` of the module `psumlac.mod` implements the real addition algorithm. The `step` parameter, assigned in the IAC level, specifies the shift length for each addition.

psumlac.mod continues ...

```
PROCEDURE ComputeSums(step: ShiftDistance);
BEGIN
  Declaration(sf{0..15}, sf{32..47}, sf{});
  Reset(Inc, ro{p1, m1, p2, m2});
  Uncond(TagBit(bm{}, sd{}, ab{}, tr1));
  ResetCarry(NoClr, slOpen, a);
  WriteSegLinks(s64);
  Uncond(TagBit(bm{}, sd{}, ab{}, tr1));
  BitWrite(NoClr, slOpen, a, bm{}, sd{}, ab{a10});
  Uncond(TagBit(bm{}, sd{}, ab{}, tr1));
  BitWrite(NoClr, slOpen, a, bm{}, sd{}, ab{a30});
  Uncond(TagBit(bm{}, sd{}, ab{}, tr1));
  TagShift(slOpen, step);
  BitWrite(NoClr, slOpen, a, bm{}, sd{}, ab{a31});
REPEAT

 (* copy bit*)
```

```
  IF TagBit(bm{}, sd{f11}, ab{}, tr1) THEN
    TagShift(slOpen, step);
  END;
  BitWrite(NoClr, slOpen, a, bm{}, sd{f21}, ab{});
  BitWrite(NoClr, slOpen, am,bm{}, sd{f20}, ab{});

(* addition*)

  Uncond(AddTag(sf1, sf2, ab{a31}));
  BitWrite(NoClr, slOpen, a, bm{}, sd{f11}, ab{a11});
  Uncond(TagBit(bm{}, sd{}, ab{a10, a31}, tr1));
  BitWrite(NoClr, slOpen, a, bm{}, sd{f10}, ab{});

(* reset activity bits *)

  Uncond(TagBit(bm{}, sd{}, ab{a31}, tr1));
  BitWrite(NoClr, slOpen, a, bm{}, sd{}, ab{a10});
  UNTIL CondResetIndex(Inc, ro{p1, m1, p2, m2}, xo{p1, p2});
END ComputeSums;

END psumlac.
```

% textedit psumiac.def

```
(*$$ IntermediateLevel ASP *) DEFINITION MODULE psumiac;
FROM SharedStorage
IMPORT tSharedMemAdd;
(*$$ EXPORT CROSS LEVEL InOutBufSize,
  InOutBuf,
  InOutBufPtr,
  pSums; *)
CONST
  InOutBufSize = 64;
TYPE
```

```
   InOutBuf = ARRAY [1..InOutBufSize] OF CARDINAL;
   InOutBufPtr = POINTER TO InOutBuf;
PROCEDURE pSums (buf: tSharedMemAdd);
END psumiac.
```

The IAC module exports the size and the type of the shared buffer. These definitions can be used in the HAC level for the allocation of the shared area. The buffer format will be the same in both IAC and HAC levels. The procedure `pSums` has a parameter which will contain the pointer to the shared buffer allocated.

% textedit psmiac.mod

```
(*$$ IntermediateLevel ASP *) IMPLEMENTATION MODULE psumiac;

FROM psumlac
   IMPORT
     InitApeArray,
     LoadApeArray,
     DumpApeArray,
     ComputeSums;
FROM SharedStorage
   IMPORT tSharedMemAdd;
FROM lamTypes
   IMPORT ShiftDistance, CtrlParameter;
FROM sdbHandler
   IMPORT sdbReadBlock, sdbWriteItem, sdbWriteBlock, tSdbItemType;
FROM xpcAndDmaStatus
    IMPORT tPid, Schedule, Wait;
```

psumiac.mod continues .....

```
PROCEDURE pSums (buf: tSharedMemAdd);
VAR pid, dmapid: tPid;
    bufPtr: InOutBufPtr;
   step: CARDINAL;
BEGIN
(* Convert the Pointer *)
```

```
  bufPtr := InOutBufPtr(buf);
(* Load the Buffer into ASP *)
  Schedule(LoadApeArray (InOutBufSize), pid );
  dmapid := sdbWriteBlock (bufPtr, InOutBufSize, BinaryWord);
  Wait (pid);
(* Compute Sums *)
  step := 1;
  REPEAT
    Schedule (ComputeSums (step), pid);
    Wait (pid);
    step := step*2 ;
  UNTIL (step = 64);
(* Dump result Buffer *)
  Schedule(DumpApeArray (InOutBufSize), pid);
  dmapid := sdbReadBlock (bufPtr, InOutBufSize, BinaryWord);
  Wait (pid);
END pSums;

END psumiac.
```

The procedure `pSums` converts the pointer to the shared memory into the I/O buffer format and requests a transfer into the `swFifo` for loading the buffer into the ASP. Notice that before the loading operation, the system schedules the execution of the `LoadApeArray` routine. In this way, the LAC starts and waits for the first item coming from the IAC.

After the loading procedure synchronization, the `pSums` procedure schedules the LAC `ComputeSums` routine several times. Notice the difference between this two-level implementation and the one-level implementation used for the VASP simulator. The REPEAT loop has been moved to the IAC level.

In order to get the result, `pSums` schedules the `DumApeArray` routine on the LAC and reads the buffer coming from the LAC output fifo `rdFifo`.

Notice that the variables `dmapid` and `pid` are of the same type. This explains the different library module used for the IAC level (`xpcAndDmaStatus`) which can manage the DMA transfer with the same primitives used for the Cross Procedure Calls.

```
% textedit pshac.mod

(*$$ HighLevel ASP *) MODULE psumhac;
```

```
FROM psumiac
    IMPORT InOutBufSize,
           InOutBuf,
           InOutBufPtr,
           pSums;
FROM xpcStatus
    IMPORT Wait;
FROM SharedStorage
    IMPORT DEALLOCATE, ALLOCATE, tSharedMemAdd;
VAR
    buf:InOutBufPtr;
BEGIN
(* Allocate Shared Area *)
ALLOCATE (buf, InOutBufSize * 4);
(* Initialisation should be here*)
(* Execute on ASP *)
Wait (pSums (buf));
(* Free the Shared Data buffer *)
DEALLOCATE (buf, InOutBufSize * 4): END psumhac.
```

The module body allocates the shared area, initializes the array and calls the `pSums` routine for the addition.

In order to compile the whole application, use the following `makefile` and run the compilation with the following command:

```
% make all
```

A.2.4 The pSums makefile

The following listing is the `makefile` associated to the `pSums` application for compiling and linking each level in the right order and generating the executable target `psumhac`.

```
IACCFLAGS = -C
HACCFLAGS = -C
IACCOMP = aspc $(IACCFLAGS)
HACCOMP = aspc $(HACCFLAGS)
LINK = aspl
```

```
HACTARGET = psumhac
IACTARGET = psumiac
LACTARGET = psumlac
all: $(HACTARGET)
$(HACTARGET):\
        $(HACTARGET).o\
        $(IACTARGET).X\
        $(LACTARGET).lxb
        $(LINK) -o $(HACTARGET) $(HACTARGET)
$(HACTARGET).o:\
        $(HACTARGET).mod\
        $(IACTARGET).syx
        $(HACCOMP) $(HACTARGET).mod
$(IACTARGET).X:\
        $(IACTARGET).mod\
        $(IACTARGET).x\
        $(LACTARGET).x
        $(IACCOMP) $(IACTARGET).mod
$(IACTARGET).syx $(IACTARGET).o $(IACTARGET).x:\
        $(IACTARGET).def\
        $(LACTARGET).x
        $(IACCOMP) $(IACTARGET).def
$(LACTARGET).lxb:\
        $(LACTARGET).mod\
        $(LACTARGET).x
        $(IACCOMP) $(LACTARGET).mod
$(LACTARGET).xsym $(LACTARGET).X $(LACTARGET).x:\
        $(LACTARGET).def
        $(IACCOMP) $(LACTARGET).def
```

References

[1] R.M. Lea, ASP: a cost-effective parallel microcomputer, IEEE Micro, Oct. 1988.

[2] The MPPC Proposal, CERN/EF/MPPC 89–1 (1989).

[3] Status Report 1990, The MPPC Project, CERN/DRDC 90–76 (1991).

[4] F. Rohrbach, The MPPC Project (Massively Parallel Processing Collaboration); status and first results, *in* Proc. Int. Conf. on Computing in High Energy Physics, Tsukuba, Japan, 1991, Eds. Y. Watase and F. Abe (Frontiers Science Series No. 3, 1991), (FSS-3), ISSN 0915-8502, p. 153, and CERN/ECP 91–9 (1991).

[5] F. Rohrbach, Associative string processors in high-energy physics detectors, *in* Proc. 18th Workshop of the INFN Eloïsatron Project, Image Processing for Future High Energy Physics Detectors, Erice, Italy, 1991, Ed. V. Buzuloiu (World Scientific, Singapore).

[6] F. Rohrbach, Associative string processors for online analysis in future high energy physics experiments, *in* Applications of Digital Signal Processing to Communications, COST 229, WG4 Workshop, Leysin, Switzerland, 1992 (EPFL, Lausanne).

[7] F. Rohrbach, Massively parallel processing: a need and a powerful tool in sciences for the future, with particular emphasis on associative string processors for high energy physics experiments, Fifth Asia Pacific Physics Conference, Awana Golf & Country Club, Genting Highlands, Malaysia, 1992, and CERN/ECP 92–19, MPPC 92–31 (1992).

[8] F. G. Friedman and R.M. Lea, Radiation-hard associative string processors—a high density scalable SIMD architecture, *in* Proc. Int. Conf. on Computing in High Energy Physics, Tsukuba, Japan, 1991, Eds. Y. Watase and F. Abe (Frontiers Science Series No. 3, 1991), (FSS-3), ISSN 0915-8502, p. 223.

[9] SVIC 7213 SBus to VIC/VMV Interface User's Manual, Version 1.0 (Creative Electronics System SA, CH 1213 Petit Lancy, Switzerland 1992).

[10] VIC 8250 VMV to VME One Slot Interface, User's Manual, Version 3.0 (Creative Electronics System SA, CH 1213 Petit Lancy, Switzerland, 1992).

[11] FIC 8232 Fast Intelligent Controller User's Manual, Version 3.0 (Creative Electronics System SA, CH 1213 Petit Lancy, Switzerland, 1992).

[12] H. Le Provost, ASPEN: un contrôleur optimisé pour les applications temps réel du processeur massivement parallèle ASP (Associative String Processor), Diplôme d'ingénieur C.N.A.M., Paris, 28 avril 1993.

[13] GPM, Gardens Point Modula-2 for Sun (A+L AG Software Development System, Däderiz 61, 2540 Grenchen, CH 1992).

[14] ACE Associated Computer Experts bv, Vol. 1, COMP compiler manuals, Release 920.1 (ACE, Van Eeghenstraat 100, 1071 GL Amsterdam, 1992).

[15] ACE Associated Computer Experts bv, Vol. 2, COMP compiler documentation, Release 920 (ACE, Van Eeghenstraat 100, 1071 GL Amsterdam, 1992).

[16] User's Manual for Release 2.0 & 3.0 of the LAM Compiler Version 1.2, 1st ed., February 1992 (ASPEX Microsystems Ltd., Brunel University, Uxbridge, Middlesex UB8 3PH, United Kingdom).

[17] MPPC LAM Programmer's Reference Manual, 1st ed., March 1991 (ASPEX Microsystems Ltd., Brunel University, Uxbridge, Middlesex UB8 3PH, United Kingdom).

[18] User's Manual for the ASPC Version 1.1 and ASPL Version 1.1, Sun-3 and Sun-4 (SPARCstation) Release 2.0 of the ASP Compiler Version 1.0 and ASP Linker Version 1.0, 1st ed., March 1992 (ASPEX Microsystems Ltd., Brunel University, Uxbridge, Middlesex UB8 3PH, United Kingdom).

[19] ASTRA Application Programmer's Reference Manual, 1st ed., June 1993 (ASPEX Microsystems Ltd., Brunel University, Uxbridge, Middlesex UB8 3PH, United Kingdom).

[20] User's Manual for Release 2.0 and 3.0 of the VASP-SIM Simulator Version 1.2, rev. November 1990 (ASPEX Microsystems Ltd., Brunel University, Uxbridge, Middlesex UB8 3PH, United Kingdom).

[21] VASP-SIM Procedure Library User's Manual for Version 1 of the ASP Programming Support Package, rev. November 1990 (ASPEX Microsystems Ltd., Brunel University, Uxbridge, Middlesex UB8 3PH, United Kingdom).

[22] VASP-SIM Procedure Library User's Manual for Version 1 of the Advanced Arithmetic Package, rev. November 1990 (ASPEX Microsystems Ltd., Brunel University, Uxbridge, Middlesex UB8 3PH, United Kingdom).

[23] C.D Moffat and L.B. Orsini, ASP Cook Book: An introduction to the Associative String Processor, MPPC Collaboration, CERN/ECP/MPPC 92–30 (1992).

[24] L. Orsini, Writing ASPA Applications: Programmer's Guide, MPPC Collaboration, CERN/ECP/MPPC 93–34 (1993).

[25] L. Orsini, CERN-ASPA Machine Installation Guide, MPPC Collaboration, CERN/ECP/MPPC 93–35 (1993).

[26] C. Moffat and L. Orsini, MPPC Collaboration Software Development: Graphic Tools for the ASP machine, MPPC Collaboration, CERN/ECP/MPPC 92–27 (1992).

[27] C. Moffat, L. Orsini and F. Rohrbach, CERN-ASPA Machine User's Book, MPPC Collaboration, CERN/ECP/RA1/MPPC 93–36 (1993).

[28] ASTRA, ASP Support Team, ASPEX Microsystems Ltd., Brunel University, Uxbridge, Middlesex UB8 3PH, United Kingdom; E-mail: asp.support@brunel.ac.uk; Fax: +44 895 258728; Phone: +44 895 274000 ext. 2368.

[29] THOMSON-TMS, 50 rue J.P. Timbaud, BP 330, F-92402 Courbevoie CEDEX, France.

[30] G. Odor, F. Rohrbach, and G. Vesztergombi, Second-level muon trigger concept for the LHC, *in* Proc. LHC Workshop, Aachen, 1990, CERN 90–10, ECFA 90–133, Vol. III, p. 136. Also issued as CERN/ECP 90–20/MPPC 90–9 (1990).

[31] EAST (RD11) Proposal, and Status Report, CERN/DRDC 90–56 (30 Oct. 1990) and CERN/DRDC 92–11 (3 March 1992).

[32] Algorithm and data definition: EAST note 91–10, Benchmarking architectures with Spacal data (J. Badier, R.K. Bock, C. Charlot, and I. Legrand) (25 Nov. 1991); Algorithm and data definition: EAST note 91–11, Benchmarking with data from the Transition Radiation Detector (P. Bialas, J. Chwastowski, P. Malecki, and A. Sobala) (2 Dec. 1991).

[33] J. Badier et al., Evaluating parallel architectures for two real-time applications with 100 kHz repetition rate, IEEE Trans. Nucl. Sci. **40** (1993) 45.

[34] Benchmark Results Workshop, 11–12 May 1992, EAST note 92–16 and accompanying notes:
A. Thielmann, The ASP Benchmarks for the second-level Trigger (TRD), EAST note 92–12 (1992);
G. Vesztergombi and G. Odor, ASP algorithm for second-level TRD triggering, EAST note 92–14 (1992).

[35] D. Gabor, Theory of communication, Proc. Inst. Electr. Eng. **93** (1946) 429–57.

[36] S. Marcelja, Mathematical description of the response of simple cells, J. Opt. Soc. Am. **70** (1980) 1297–1300.

[37] T. Ebrahimi, Perceptually derived localised linear operators: application to image sequence compression, Ph.D. thesis, Swiss Federal Institute of Technology, Lausanne, 1992.

[38] F. Dufaux, T. Ebrahimi, and M. Kunt, A massively parallel implementation for real-time Gabor decomposition, *in* SPIE Proc. Visual Communications and Image Processing '91, Boston, MA, **1606** (1991) 851–64.

[39] F. Dufaux and M. Kunt, A massively parallel implementation for pyramidal Gabor decomposition, *in* Workshop on Massively Parallel Computing, Leysin, Switzerland, March 1992.

[40] G.W. Cottrell, P. Munro, and D. Zipser, Image compression by back propagation: an example of extensional programming, Technical Report ICS 87–02, ICS-UCSD, San Diego, CA (1987).

[41] S. Carrato, A. Premoli, and G.L. Sicuranza, Linear and nonlinear neural networks for image compression, *in* Proc. Int. Conf. on Digital Signal Processing, Florence, Italy, 1991.

[42] A. Basso, W. Li, A. Nicoulin, and M. Kunt, Side information compression in subband coding of video, *in* Proc. ECCV 92, Paris, September 1992.

[43] N. Sonehara, M. Kawato, S. Miyake, and K. Nakane, Image data compression using a neural network model, *in* Proc. IJCNN, pp. II35–II41 (1989).

[44] H. Boulard and Y. Kamp, Auto-association by multilayer perceptrons and singular value decomposition, Biol. Cybern. **59** (1988) 291–94.

[45] P. Burrascano, A multilayer perceptron in the Chebyshev norm for image data compression, *in* Proc. IEEE Symp. on Circuits and Systems, 1991.

[46] E. Augé and A. Ducorps, First application of a massively parallel system for on-line processing in an HEP experiment, LAL Report RT/93–02 (1993).

[47] Microware Systems Corp., Des Moines, IA, USA.

[48] J. C. Brisson, P. Le Dû, and B. Thooris, Prospect to use massively parallel processors in the SDC second level trigger, *in* Proc. Int. Conf. on Computing in High Energy Physics, Tsukuba, Japan, 1991, Eds. Y. Watase and F. Abe (Frontiers Science Series No. 3, 1991), (FSS-3), ISSN 0915-8502, p. 165.

[49] S. Zylberajch, MACHOS, WIMPS or Dust: what is dark matter?, CEA-CEN Saclay Report DPhPE 91–02 (1991).

[50] D. Calvet, Evaluation du calculateur parallèle ASP, application à l'expérience des Naines Brunes, CEA-CEN Saclay, rapport de stage à Saclay, 23 juillet 1991.

[51] A. Sandoval, Fine grain parallel processor, *in* The LAA Project, A. Zichichi, CERN/LAA 89–1 (1989), II.6.c., p. 286.

[52] NA35 Collaboration, G. Vesztergombi et al., 'Iconic' tracking algorithms for high energy physics using the TRAX-1 massively parallel processor, *in* Proc. Conf. on Computing in High-Energy Physics, Oxford, 1989, Comput. Phys. Commun. **57** (1989) 290–96.

[53] A. Ster, WA93 image processing application on the ASP machine, CERN/ECP/MPPC 92–29 (1992).

Sic itur ad ASTRA!