HAPI : a Hardwired Array Processor Interface for a real time image processing machine[1]

Serge WENDLING

Université Louis Pasteur, ENSPS / LSIT
7, rue de l'Université 67000 STRASBOURG

RÉSUMÉ

Dans une première partie nous présentons les principales caractéristiques de MASYVE, une machine "hybride" pour le traitement d'images en temps réel vidéo développée dans notre laboratoire. Nous considérons principalement le problème des échanges de données entre processeurs, goulot d'étranglement de nombreux systèmes. Nous décrivons plus en détail un processeur d'E/S câblé mais programmable et montrons comment il permet de contrôler les très importants flots de données circulant dans le système.

ABSTRACT

This paper first exposes the main characteristics of an "hybride" SIMD / MIMD machine called MASYVE and built in our laboratory for real time image processing. It focuses on the problem of the data exchanges which is very often neglected though it becomes the bottleneck of many systems. A hardwired but programmable I/O processor is described with some details and it is shown how such an interface can be used to efficiently control the very important datae flowing through the system.

Introduction

Parallel machines make it possible to reach the computing power requested in real time image processing. But calculating very fastly is not enough. There must also exist a mechanism managing the very important data sets flowing through the processors. If such a facility is not provided, the I/Os will become the bottleneck of the system. In this paper, we describe a programmable input/ output interface which synchronizes several processing units on their data flows. A first part will introduce the architecture of MASYVE, an image processing machine built in our laboratory and consisting of two hardwired I/O processors and three processing units : a data acquisition and preprocessing workstation (ICOTECH), an array of 48 by 48 quasi systolic processors (GAPP) and up to four array processors (ZIP) [PHE+88],[EDH+89],[PHE+89]. We then focus on the array processors and give some details about HAPI (Hardwired Array Processor Interface) which implements a wired "producer-multiconsumers" scheme where the producer and the consumer(s) can be changed dynamically at any time by software. Finally, we will introduce some possible topologies for the MASYVE machine which are made possible by HAPI (data flow sharing and/or program flow sharing), the final goal of our approach being to find out how to best use each processor for optimizing the throughput of the machine for a given sequence of algorithms.

The MASYVE machine

The developpment of systems able to execute image processing algorithms very fastly can be considered in two ways: the first one leads to develop dedicated hardwired processors assuming each one a specific function and statically "re-connected" together each time a given sequence of processes is to be performed. Such an approach makes it possible to meet real time performances but has at least two main drawbacks : its costs and its poor flexibility. The second one, considering that most algorithms encountered in image processing do fit in only a few classes for which there exists some optimal structure of parallelism, leads to the design of "hybrid" machines, still satisfying the "near real time" constraints but being fully programmable and dynamically configurable by software ("near real time" stands for systems which do not always work at real video rate. For some algorithms, they spend several frames to compute the results. In many situations, this is not really a drawback. Even more, for numerous applications, "operator real time", which means an answer within one or two seconds, is almost fast enough). Considering this last possibility, we developped the MASYVE machine shown in figure 1 and consisting of :

- ICOTECH, built in our laboratory around a standard 16 bit SISD processor and used to digitize or display images and to perform at video rate some very low level algorithms such as image summing or averaging, profiles, ...[DZW86].
- A GAPP machine (Geometric Arithmetic Parallel Processor) commercially available from NCR and built around 32 chips, containing 6*12 Processing Elements each, to form an array of 48 by 48 bit serial processors. This SIMD structure is very efficient for local processing such as integer zooming, thresholding, edge detection or N by M spatial filtering (for each processed pixel, these algorithms need only knowledge about a limited neighbourhood) [HER88].
- Up to four array processors (ZIP) well suited for medium level algorithms such as global transformations (FFT, transform, convolution and correlation evaluations), image zooming or rotating by any real factor or angle and with bilinear interpolation, conversion of a pixel structure into list structures...
- A host processor (in our case a PC) managing the whole system (through a BIT 3 to Multibus I interface card) in loading the object codes into the other processors, configuring the data pathes and interfaces, sequencing the execution of the different tasks, providing a convivial interface between the user and the machine.

[1] This work is part of a project partially supported by the Commissions of the European Community under the contract ESPRIT P26.

- A four chanel numeric video bus with a transfer rate up to 160 Mb/s and a specific interface for the GAPP machine. This interface, called the "windower" [EUG89], performs the I/Os for the processor array and formats its data according to the size of the input image and the window size processed by the GAPP. It is software configurable and resolves the problem of overlapping regions in the case of local processing (because the GAPP array size is much smaller than an image, it is necessary when implementing an operation involving an N by M neighbourhood, to slice the input image into sub-images. To avoid erroneous results near the borders of these sub-images, the windower feeds the GAPP with a set of (N-1)/2 by (M-1)/2 overlapping regions and re-arranges the results).
- The HAPI interface which integrates the array processors into the rest of the system as will be shown below.

The array processors : MERCURY ZIP 3216

These commercially available processors have been choosen because they provide following features :
- a standard BUS : at the time they were considered, the main part of the system (ICOTECH and GAPP) did already exist and used a Multibus I,
- compatibility with the transfer rate imposed by the existing numeric video bus (one 8 bit pixel every 65 ns),
- local memory for the programs in each of the two processors building a ZIP which speeds up the performances in limiting main memory access conflicts,
- numerous I/O possibilities (Zip Bus, Multibus, auxiliary ports) and user defined control and status lines which facilitate the integration of the ZIPs into a system and makes it even possible to build a "ZIP network",
- a wide software environnment ("C like" compiler, image processing library, simulator for debugging).

The general architecture of a ZIP 3216, as shown shown in figure 2, includes :
- a control processor (CP) assuming the I/Os, feeding the arithmetic processor with data and saving its results, performing main storage accesses and processing interrupts. It is a scalar processor with 32 internal registers and 16 Mbytes addressing capabilities.
- an arithmetic processor (AP) assuming data in fixed point format (block floating point).
- a 32 bit wide internal bus (ZIP Bus) with a transfer rate up to 40 Megabytes/s,
- an interface to a host system with DMA possibilities,
- two auxiliary ports for I/Os with peripherals which were used to integrate our HAPI interface.

Each ZIP is clocked at 10 MHZ and most instructions are executed within one clock period (100ns).

The HAPI interface

Many authors have described machines with quite astonishing fast computing possibilities. Unfortunately, when they speek about performances, they most often suppose that the code of the algorithms to be executed and the data to be worked on are already in core. Such an approach leads to give an erroneous idea of the true possibilities of a machine because the important point is not, say, to perform a 512 by 512 real FFT in a few hundreds of milliseconds, but to know how long it takes to process completely an image (or set of images), starting from its capture and ending with the display of the final searched for results. For many applications this means that the performance of a machine is to be evaluated as its global throughput, that is finally, the number of images it can process within a given time slice. Thus, when considering the execution of a sequence of tasks onto several processors, the I/Os needed to load the different processors with their object code or to transfer data or results from one processor to the other, can become a severe limitation canceling all the benefits of ultra fast CPUs. To take into account this constraints, we devoted much efforts to build efficient interfaces around the MASYVE machine. Concerning the ZIP processors, three directions have been investigated :
- Multibus I/O : it allows data transfers either under program control or by direct memory acces (DMA). Unfortunately its transfer rate is by far too low to make it usable for such an important data flow as the one imposed by real time image processing. However, it will be used to configure the system (only a few bytes are needed) and to download the code of the algorithms to be executed into the different processors (due to their large local memory, this is to be done only once).
- ZIP bus I/O : as it allows a transfer rate up to 40 MB/s, it seems to be the best solution for video real time exchanges. However we discarded it for two reasons : 1) we would have had to build a " ZIP bus interface " for all the other processors (ICOTECH and GAPP), 2) we wanted to keep an open system in which a component can be changed without having to rebuild the whole machine.
- auxiliary ports I/O : these two 16 bits wide ports (one for input and one for output) allow data exchanges in a similar way as does a Centronics port for a printer (the main difference however is that the ZIP always works as an "answering device", that is, for each exchanged word, in any direction, it is the external device which has to initiate the transfer). Maximal transfer rate is 5 MHZ that is, at least 200 ns for two bytes. This is obviously not fast enough for our purpose because ICOTECH sends or needs at least one byte every 65 ns (there is no way to slow ICOTECH down because it is sequenced by the video clock and works on a "burst mode" principle). However, interesting features are offered : 1) the two 16 bits one directional ports can be tied together to build a single bidirectional 32 bits I/O port which theoretically allows a transfer of 4 bytes in 200 ns (versus 260 ns for ICOTECH), 2) the reading from (or writing to) the ZIP ports can be initiated by external synchronization signals, 3) some command and status I/O lines are left free for the user.

Consequently, the design of an interface satisfying the video real time requirements is possible even if non trivial timing problems have still to be solved: 1) all the I/Os have to be synchronized with ICOTECH which imposes its clock and control signals, 2) ICOTECH cannot send or receive less than one line at a time which implies 512 bytes during each active part of a video line, 3) the time of 200 ns for a 4 bytes exchange on the ZIP side is a "at best" transfer rate which cannot always be satisfied because of periodic memory refresh interrupts (every 96 µs).

All these considerations, lead to the HAPI interface schematized in figure 3. It can support up to four ZIP array processors, the GAPP and the ICOTECH machines. It basically realises a programmable wired "producer-multiconsumers" scheme with data reformating. The use of FIFOs which can be read and written asynchronously and at the same time, resolves the problem of speed differencies between the processors.

A data exchange involves always a unique source (either ICOTECH, or GAPP or one of the ZIPs) and one or more destinators. Following table summarizes the different possible

data flows (as already stated, the I/Os between ICOTECH and GAPP are performed by another interface called the "windower" which is described in [EUG89]) :

SOURCE		DESTINATOR(S)	
Processor	**Format**	**Processor(s)**	**Format**
ICOTECH	8 bits	1 to 4 ZIP	32 bits
GAPP	8 bits	1 to 4 ZIP	32 bits
ZIP	32 bits	ICOTECH	8 bits
ZIP	32 bits	GAPP	8 bits
ZIP	32 bits	1 to 3 ZIP	32 bits

The ability to have several ZIPs as destinators makes it possible, for instance, to fetch an image from ICOTECH (or a result from GAPP) and send it at the same time to all the ZIPs which can keep it in its own or store only a part of it. Thus, it is possible to feed efficiently the processors with data for several different topologies of the system (data partitioning or algorithms partitioning). Another point to be noted is that the interface does never know the size of the exchanged blocks which simplifies both its hardware realization and its programming, and reduces the system overhead to configure the interface.

A transfer is always initiated (but not controled) by a ZIP and starts as soon as all the concerned processors assert their ready control line (the processors not involved in the exchange just carry on their computing, concurrently with the I/O). Synchronization of the exchanges and generation of all the needed control signals are performed by the interface which leads to very short I/O software for the ZIPs, the only way to reach the expected response times (about 200 ns for 4 bytes, that is 160 Mbits/s). Programming HAPI is quite easy and consists in defining, for each transfer, a control word specifying the source and the destinator(s). The succesive commands needed to carry out a sequence of algorithms are generated by the application program running onto the host processor and in full transparency to the end user. Figure 4 shows how HAPI communicates with the host which uses the Multibus to gain access to :

- a command queue (FIFO) executed sequentially by HAPI (each command corresponds to a transfer of one up to 512 lines of 512 bytes each),
- an instruction register holding the command in progress,
- a status register showing the state of the interface,
- two control lines (START and STOP) making it possible for the host to suspend and resume commands execution, which is very helpful for debuging purposes.

Using a command queue and defining two instruction modifiers (the LOOP bit and the PAUSE bit) is very interesting when a sequence of algorithms is to be performed repetitively. In such a case, the I/O commands have to be downloaded only once because HAPI restarts automatically with the first command in the queue if the LOOP bit is set in an instruction (similarly, when the PAUSE bit is set, execution is suspended until a START signal is sent to the interface by the host). Thus, in many applications (e.g. scene analysis with features extraction to guide a robot), it is possible to initialize the interface and "simply forget it".

An application running onto MASYVE consists of a sequence of tasks called by the host processor. Each task is executed by one (or more) processor(s) (host, ICOTECH, GAPP, ZIPs) and can be decomposed into three steps : 1) acquisition of the data to be processed, 2) execution of one (or more) algorithm(s) onto that data, 3) restitution of the computed results. Concerning the ZIPs, all the I/Os are performed by HAPI and are initiated by calling specific procedures with their correponding arguments. Such a procedure which leads to synchonize the different processors onto their data flow to reach the best global throughput for the machine.

Beacuse there is no global sequencing clock, each component of the system works asynchronously to the others and at its maximal speed. Thus, it becomes possible to improve the performances of the machine in simply replacing one given element with a more powerful one, without having to reconsider the others (the integration of two 386 processor-cards working at 33 Mhz and with 387 co-processors, is currently being realized. These cards will be seen by HAPI as two ZIP processors and lead to more flexibility, less expense, better performances and considerable simplifications in algorithms programming).

Conclusion

The MASYVE machine and its interfaces have been built to study the advantages of using concurrently different processor architectures in image processing (figure 5 shows some possible configurations for our system). The results obtained until now confirm the interest of our approach but bring also into light the difficulty of deciding which processor (or which set of processors) is (are) to be used for each algorithm of a given sequence to optimize globally the response time of the system. We do think that a possible way to solve this problem is to specify characteristics which apply to the different processors independently of their internal structure, that is taking into account only parameters such as loading times for object code, transfer times for data and execution times for algorithms. A correct evaluation of these parameters for different configurations of the system, and the fact that in most image processing applications the sequence of algorithms to be executed is known a priori and applied repetitively, should then reduce the problem to a simple valuated graph in which a "optimal way" is to be found. Once obtained, this optimal path will bring an answer to the crucial question "**when** do we best use **which** processor(s) to achieve the fastest throughput?" The formulation of this approach is currently under study and the MASYVE machine will serve as a tool to verify our assumptions [SW].

Acknowledgements

The author whishes to thank to the people working upon the ESPRIT project P26 for their help and kindful permission to use some of the illustrations presented in this paper. He especially thinks of C. DRAMAN, R. EUGENE, Y. HERVE, E. HIRSCH and F. PIERRE.

References

[DZW86] C. DRAMAN, K. ZAMPIERRI, P.L. WENDEL, "Poste de traitement d'images configurable : icotech", Semaine internationale de l'image électronique de Nice, 1986.

[HER88] Y. HERVE, "Mise en oeuvre et optimisation de l'utilisation d'un processeur quasi-systolique dans une chaîne de traitement d'images temps réel", Thèse de l'Université de Strasbourg, april 1988.

[PHE+88] F. PIERRE, Y. HERVE, R. EUGENE, C. DRAMAN, S. WENDLING, "Description d'une machine de traitement d'images temps réel: l'approche multi- parallélisme", PIXIM Paris, 1988

[EDH+89] R. EUGENE, C. DRAMAN, Y. HERVE, F. PIERRE, S. WENDLING, "Architecture multi-parallèle : Applications en traitement d'images", GRETSI, Juan le Pins, april 1989.

[PHE+89] F. PIERRE, Y. HERVE, R. EUGENE, C. DRAMAN, S. WENDLING, "A machine for video real time image processing using parallel polymorphism", CA-DSP 89, Hong Kong 1989.

[CAET88] CSELT, AEG TELEFUNKEN, ENSPS, THOMSON CSF, "Advanced Algorithms and Architecture for Speech and Image Processing", Esprit Project P26, final report September1988.

[SW] S.WENDLING "Contribution à l'architecture des ordinateurs et au traitement d'images. Mise en oeuvre d'une structure parallèle hybride par une recherche d'optimisation globale", Thèse d'Etat, Université de Strasbourg (France), January 1992.

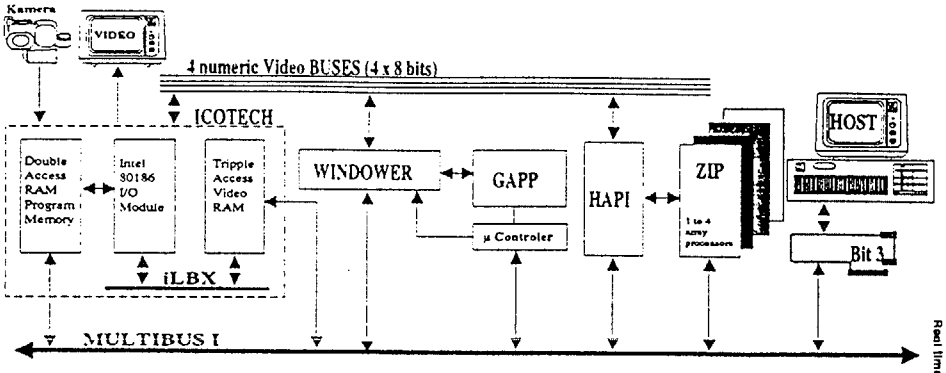

Figure 1

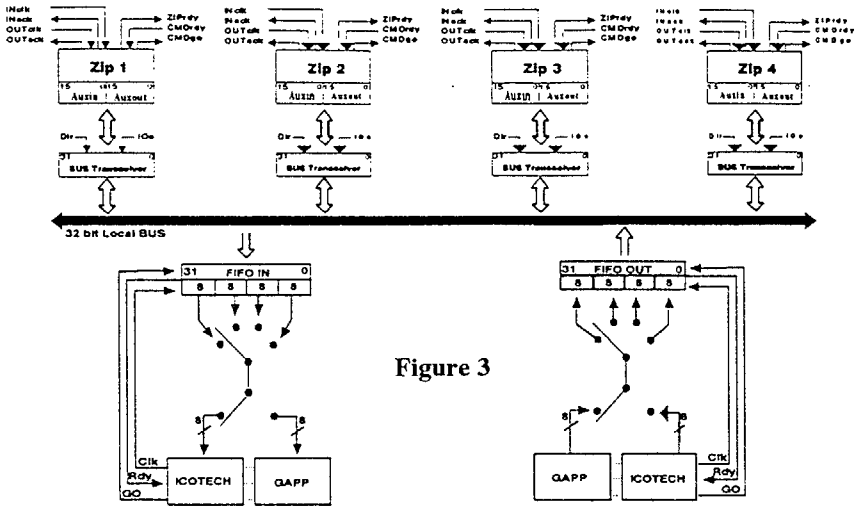

Figure 3

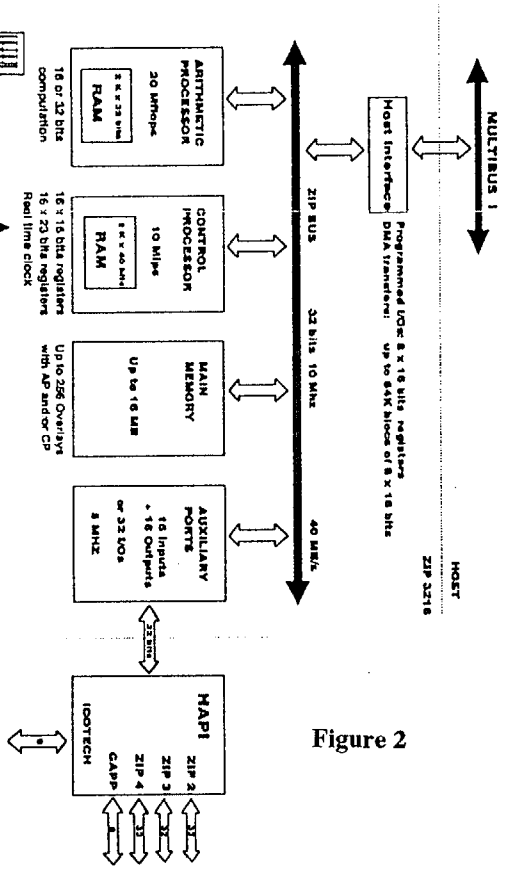

Figure 2

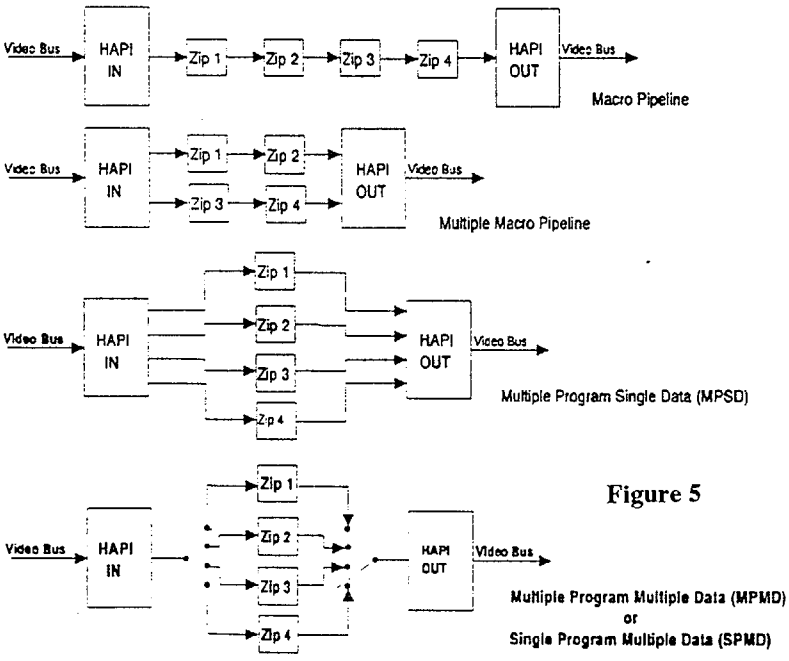

Figure 5

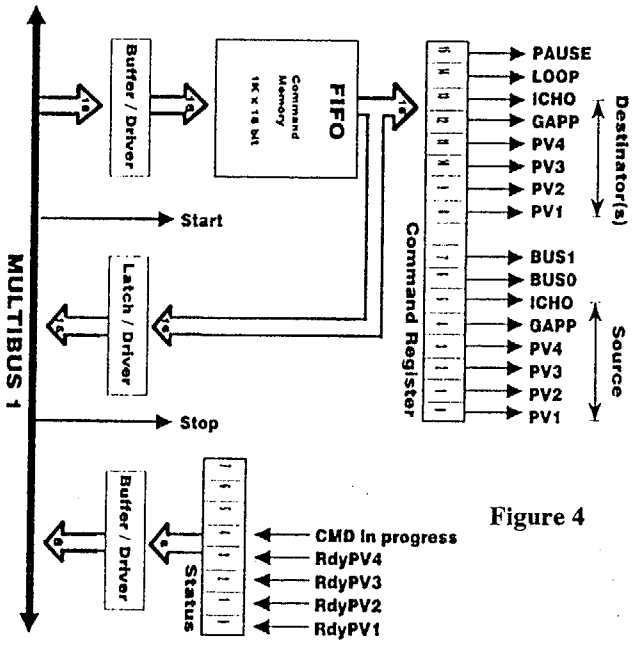

Figure 4