A SIMD SYSTEM FOR IMAGE PROCESSING *

Celso Luiz Mendes
Cecília de Azevedo Castro César
Antonio Francisco Junior
Flávio Roberto Dias Velasco

INPE - Instituto de Pesquisas Espaciais
Caixa Postal 515 - S.J.Campos, SP 12201 - Brasil

ABSTRACT

This work presents a massively parallel system that is being built at INPE, based on the GAPP device, for image processing applications. This parallel system will operate as a coprocessor of a PC-like microcomputer. After introducing the architecture, a preliminar evaluation of the system is shown, followed by a description of the software environment that is being developed. This environment includes assembly-level and high-level programming tools. Some comments on the current situation of the project are given at the end.

1. INTRODUCTION

It has long been suggested that SIMD architecture is the most appropriate organization for the implementation of low-level image processing operations (1)(2). These operations are characterized by their locality (the processing of each image point uses only the point itself and a small neighborhood) and repetitiveness (the same operation is performed all over the image). Moreover, the processing operations are usually simple and iterative. These characteristics were exploited by some proposals like the MPP (3) and DAP (4).

Although the processors can be simple and their spatial arrangement very regular (mirroring the image structure), they have to be employed in large numbers to be effective. In SIMD

* This work was supported by SID Informática ("Estra" Project) and by FAPESP (Project "Computação: 87/1786-1").

organizations for image processing, arrays of 128 by 128, comprising thousands of processors, are common. The gain in speed that can be achieved, however, can be impressive. Also, because the organization is well matched to the problem, the programming of the low-level image processing operations is not such a formidable task.

Notwithstanding the potential gains, this organization has not been more widely deployed due to the non-availability of the individual processors in the market. This situation has changed since the introduction of the GAPP (Geometric Arithmetic Parallel Processor) (5) by NCR. This made possible for research and educational institutions to experiment with several SIMD system organizations.

This work describes one of INPE's project, which objective is to develop a SIMD coprocessor based on the GAPP chip and programming environment for the IBM-PC microcomputer and compatibles. The coprocessor system is composed of a processor array, a data memory, an I/O controller and an array controller. The existence of two controllers, one specifically for I/O, makes it possible to execute I/O instructions in parallel with the processing of the individual units, giving an additional level of parallelism. The programming environment resides on the PC and is composed of a simulator, an assembly language, a linker, a preprocessor for the C language and a C-like language compiler.

In this paper, Section 2 describes the general architecture of the coprocessor. This section gives special attention to the solution that has been adopted for the I/O transfer of data and how the architecture exploits the possible parallelism between I/O and processing. An evaluation of the proposed architecture for typical image processing operations is shown in Section 3. The basic programming environment, which is composed of the simulator, the assember and the linker is briefly explained in Section 4. The high level tools for programming: the C preprocessor and the C-like compiler are described in Section 5. The final section presents the current status of the project and the perspectives for future work.

2. SYSTEM ARCHITECTURE

The architecture of the coprocessor is shown in Fig. 1. The general coordination of the system will be provided by the PC, which will also communicate with peripheral devices, interact with users and house the programming environment. For the execution of tasks in the system, the PC itself will process the sequential parts of the programs, and will activate the coprocessor for the execution of the parallel operations. This scheme is very similar to that used on the Connection Machine (6).

The coprocessor is connected to the PC through its expansion bus. The main module of the system is the array of processing elements, where parallel computations take place. It's an array of 48x48 P.E.'s, implemented with 32 GAPP chips connected simply side by side, due to the modular characteristic of this device. The edges of the array may be programmed in one of two modes: in the first one the north edge is connected to the south and the east to the west. In the second mode there is only a connection between the north and the west, providing a way to reformat data inside the array. This reformatting is a transformation between the pixel/byte access employed by the PC and the bit-plane access by the array, which is a common feature of bit-serial SIMD machines.

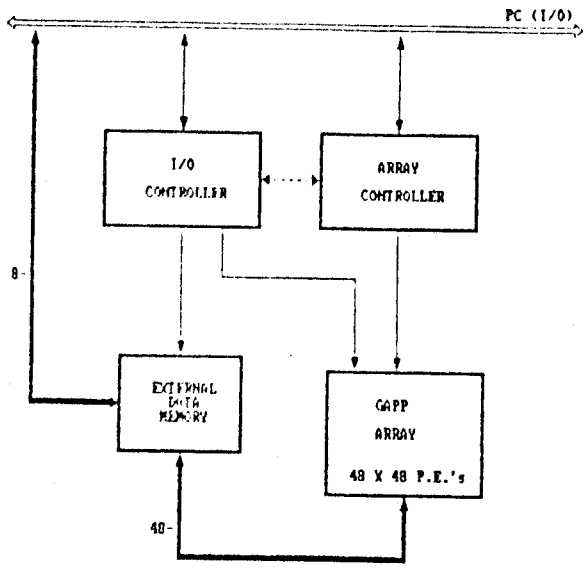

Fig. 1 - Basic architecture of the system

The image data to be processed are stored by the PC in the external data memory, on a byte sequential mode. From this memory, data may be transfered to the array, with the simultaneous transmission of 48 bits. In the maximum array speed (10 MHz), this represents an I/O rate of 60 MBytes/sec.

Operations on the coprocessor are supervised by two controllers: an array controller and an I/O controller. They are both implemented with a microprogrammable structure. Each of them has its own instruction sequencer (Amd-2910A), a fast microprogram memory, which may be loaded by the PC, and some special purpose registers. These controllers may operate independently of each other, but they are fed by the same clock signal and there is a special mechanism, represented by the dotted lines in Fig. 1, which allows the synchronization between them, if necessary (7).

The basic function of the I/O controller is to coordinate the transfers of data between the external data memory and the array of P.E.'s. This includes the flow of data through the I/O registers of the GAPP processors and also the access to the local memory inside each P.E. The microinstruction for this controller is 45 bit long.

The array controller provides the control signals that drive all operations inside the GAPP array. It has a microinstruction of 63 bits, from which are derived the GAPP instruction bits that control the ALU, and also the control lines to the special purpose registers. These registers include some local memory address registers, a global data register for broadcast operations and also an index register, to select a single bit from the global data. The mode of connection between the edges of the array is also set by this controller.

This structure was built to take advantage of the potential parallelism between I/O and processing, which is instrinsic to the GAPP chip. By having the two controllers operating simultaneously, it is possible to have general processing and flow of data through the I/O registers, in a cooperative fashion. The single contention problem happens when the I/O

controller accesses the local memory, since the P.E.'s may also be accessing that memory. The solution adopted to this problem was to stop the array controller. This does not cause much degradation in performance, as for each access of the I/O controller into local memory there are 48 corresponding shifts, to get the data in or out of the array. Thus, the worst degradation is around 2%.

If the data to be processed have just been stored by the PC on the external memory, the processing of I/O will include the reformatting of data, which shall be done on the array. If, however, the data have been previously processed on the array, they may be stored in the external memory in bit-plane mode, and so the I/O may be done simultaneously with other tasks. The current configuration for the direct connection between the external memory and the array takes this fact into account.

Although there is the extra overhead for reformatting, this configuration provides the maximum I/O rate for the case of previously formatted data. If an additional hardware had been used to speed up the reformatting process, the resulting performance would be the same for both formatted and unformatted data, and also the cost of the system would be much higher. As it is anticipated that the I/O will be frequent, due to the small size of the local memory in each P.E. (128 bits), the present configuration seems to be the best choice.

3. EVALUATION OF TYPICAL OPERATIONS

In order to evaluate this architecture, some typical image processing operations were analysed. These operations were spatial filtering on gray level images and erosion on binary images. They were chosen because, besides being time-consuming for conventional machines, they exercise several characteristics of the array, like transfer of data, arithmetical and logical operations, global transmissions, etc.

In spatial filtering, with the non-recursive case, the image is convolved with a 3x3 mask, in order to perform a given transformation (8). The characteristics of the result will

depend on the values of the mask. In the present system, comprising an array of 48x48 P.E.'s, the processing of large images requires their division in sub-images of size 48x48, with a common pixel in the edges of neighbor sub-images. For an individual sub-image, it's assumed that each pixel is stored in a corresponding P.E.. The convolution is performed by transfering data between P.E.'s, multiplying pixel values by corresponding mask values, and adding all the products together. Since the mask values are constants, they may be transmitted in a broadcast operation.

This filtering operation, when simulated on the present system, assuming images 512x512, 8 bits/pixel and a clock of 10 MHz, may be run in a total of 1,5 s. This rate is much better than what is normally obtained with the PC (520 s) and even better than the rate achieved with a special purpose board, using a TMS-320C25 digital signal processor (4 s).

In the erosion operation, the binary image is convolved with a given binary mask. For the original points that have the value 1 their neighborhood is compaired to the mask. If they match, the 1 remains; otherwise, that is, if there is any mismatch, the point becomes 0. So, the area of the objects is decreased. This operation, together with dilation, may form the basis of an image analysis system (9).

The result of simulation was a time of 18 ms for each 512x512 binary image, with a 3x3 mask. This represents a very large gain over the PC (40 s), due to the intrinsic bit-serial mode of the GAPP, where the time is linearly dependent on the number of bits/pixel. Thus, it's a very effective tool for the processing of binary images.

4. BASIC PROGRAMMING ENVIRONMENT

Some basic programming tools have been developed to aid the process of programming the system. These tools are divided in two major groups. The first group is composed of tools for the chip level. In the second group, there are tools for the whole system, including the I/O and array controllers. All these tools

have been completely developed at INPE, using the C-language on an IBM-PC under MS-DOS 3.2.

The modules of the first group are shown in Fig. 2. The main objective is to simulate an array of up to 64x64 GAPP P.E.'s. The assembler module receives instructions as documented on the GAPP data sheet, and produces the corresponding object code file, which is read by the simulator. Since there is no control structure on the GAPP itself, the user must write the program according to the desired sequence of operations. This set of tools proved to be an effective means for a basic understanding of the GAPP chip.

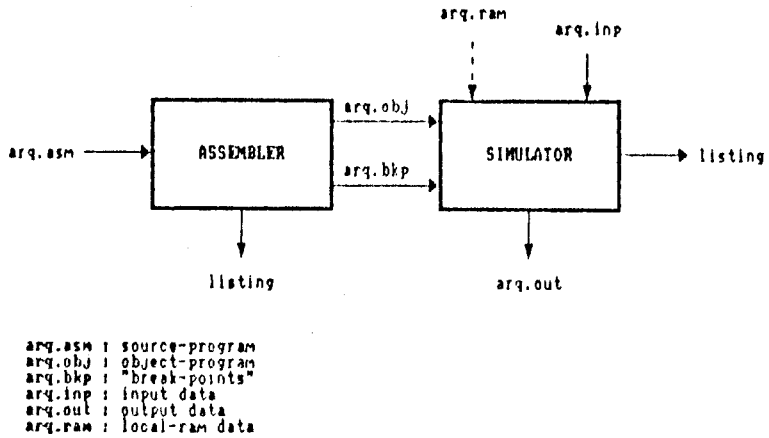

Fig. 2 - GAPP programming tools

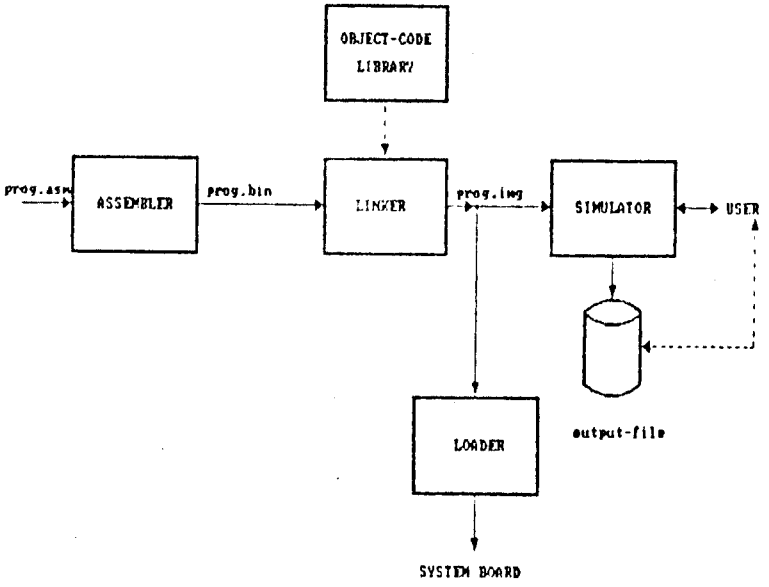

Fig. 3 - System programming environment

With the second group of tools, presented in Fig. 3, it is possible to write and simulate programs for the whole system. The programs for each controller, composed of lines with instructions for each module, may be assembled and linked individually, and then submitted together to the simulator or to the real system, when it becomes available. This simulator has many interactive features, which let the user watch or modify every register or memory in the system. Although many hardware aspects (delays, glitches,...) are not taken into account, and the simulation time is long for a common PC, this simulator is a good instrument to test the correct execution of new algorithms.

5. HIGH LEVEL PROGRAMMING

The programming of this system in the assembly level, besides being difficult, would not be reliable for medium-to-large applications. It was soon foreseen that compilers for high level languages would be needed. Moving in that direction, two developments were started: a C-like language compiler and a preprocessor for the C language.

The C-like language is dependent not only of the GAPP processor but also of the coprocessor structure. It basically enables the programmer to use high-level syntax and control structures, still having free access to all the resources of the system. The result of this compilation is a source program that can be submitted to the assembler.

The control of the execution flow is provided by the Amd sequencers, since they have instructions to implement the three basic constructions of structured programming (sequence, iteration and decision). The synchronization between controllers is implemented with the special instructions "wait" and "signal", which are converted by the compiler to one or more accesses to corresponding bits in the hardware.

Another nice feature of the language is the emulation of masking, which is not available on the GAPP chip. There is a "where" instruction that, by means of logical operations, causes the virtual blocking of access to the local memory of individual

processors, making values on the memories of masked processors remain unchanged.

The C preprocessor implements an extension of the C language for vector and array processing which is independent of the underlying architecture (including array size). The idea is to provide an almost familiar environment for the application programmers. For this preprocessor, a new language, namely CP, is being designed, including the special data qualifier PARALLEL, which will identify the objects that should be processed in a "different" way. For these objects, there is a collection of possible operations.

Each of these PARALLEL data items will be treated as an abstract data type, and the manipulations on it will be implemented according to the architecture where the program should run. So, the dependency on the real architecture is restricted to the implementation of some manipulation routines, not to the source programs. The same source program should be able to compile and run in a number of different architectures, as soon as each of them has its corresponding routines.

For the present system, PARALLEL data shall be allocated on the external data memory and be processed inside the GAPP array. For each operation involving this kind of data, the preprocessor will generate calls to routines that activate microcode procedures, which really perform the operation. If the images are larger than the array size, these routines may take advantage of the available parallelism between I/O and processing, in order to increase performance.

6. CONCLUSION

The system architecture has been designed and the assembly of a prototype should start as soon as the chips are available. Unfortunately, this has been delayed by NCR's decision of not selling the GAPP chips to INPE. In the meantime, software tools have been developed. Up to now there are two low-level programming environmets available, for the GAPP chip and for the whole system, both of them with an assembler and a simulator.

The C-like compiler has been completely specified, and implementation has just begun. The C preprocessor is being specified.

Preliminar simulation results have proven the feasibility of using this architecture in some simple image processing applications. The development of high level languages should bring an additional degree of functionality to the system. With the building of this coprocessor, massive parallelism shall be within the reach of personal computer users.

ACKNOWLEDGMENT

The authors would like to thank Dr. Edward Davis (North Carolina State University) for the original proposal of a GAPP based system and for his suggestions and comments about this implementation.

REFERENCES

1 UNGER, S. H. A Computer Oriented Toward Spatial Problems. **Proceedings of the IRE, 46**:1744-45, Oct. 1958.

2 FLYNN, M. J. Some computer organizations and their effectiveness. **IEEE Trans. Comp., C-21**(9):948-68, Sept. 1972.

3 BATCHER, Kenneth E. Design of a Massively Parallel Processor. **IEEE Trans. Comput., C-29**(9):836-40, Sept. 1980.

4 PARKINSON, D. The Distributed Array Processor (DAP). **Computer Physics Communications**, 28:325-36, 1983.

5 NCR CORPORATION, Dayton. **Geometric Arithmetic Parallel Processor**. Dayton, 1987. 12 p.

6 THINKING MACHINES CORPORATION, Cambridge. **Connection Machine Model CM-2: Technical Summary**. Cambridge, 1987.

7 MENDES, C. L. **Arquitetura Paralela para Processamento de Imagens**. ITA, São José dos Campos, SP, Master's Thesis, 1987.

8 MASCARENHAS, Nelson D. A. & VELASCO, Flávio R. D. **Processamento Digital de Imagens**. São Paulo, SP, Quarta Escola de Computação, 1984. 2 v.

9 BARRERA, Junior. **Uma Abordagem Unificada para os Problemas de Processamento de Imagens: A Morfologia Matemática**. INPE, São José dos Campos, SP, Master's Thesis, 1987.