MVA'90 IAPR Workshop on Machine Vision Applications Nov. 28-30,1990, Tokyo

Component labeling on SIMD-SPMD Architecture

P. Duclos, G. Giraudon*, P. Kaplan, M. Auguin, F. Boeri

Laboratoire Signaux et Systemes (LASSY, Universite de Nice, 41, Bld Napoleon III, 06041 Nice cedex, France
*INRIA Sophia-Antipolis, 2004 route des Lucioles, 06565 Valbonne Cedex, France
giraudon@mirsa.inria.fr

Abstract

We present a component labeling algorithm for a coarse-grained parallel architecture named OPSILA. OPSILA is an experimental general purpose multiprocessor which uses two different forms of parallelism. The first one is a well known SIMD mode (synchronous mode) and the second one is called SPMD (single Program Multiple Data stream) which is an asynchronous mode. We have shown OPSILA's effectiveness for the low-level image processing [Duc88a] and we demonstrate here its effectiveness for the intermediate level with a component labeling implementation. The aim of the implementation is obtained after processing the best load balancing efficiency. So, we propose a load distribution algorithm which incorporates a tolerance factor on the load function. This algorithm can take into consideration the initial localization of region and we show it is the better choice to the load balancing efficiency versus the cost of list transfer. Some results are given on real image.

1 Introduction

In Computer Vision, image interpretation is made roughly in three "pipelined" steps [Han78]. At each step corresponds a data structure. The Low Level vision deals with iconic structure (pixel array level). The High Level processing works with tree and graph structures (symbolic level). The Indermediate Level processing runs by taking as input matrix of pixels and creating as output lists of symbols (pixel matrix ⇒ list set). The typical algorithm of the Intermediate Level is the component labeling algorithm [Bal82]. The component labeling goal is to obtain a set of region lists from a pixel image. A region is a set of connected pixels which are homogeneous for a given criterious. In general, these lists are some representations of image features like regions, contours, lines etc...

So, parallelization of intermediate process is an important challenge in computer vision because of the mixed (iconic-symbolic) data structure in processing. Many authors have studied the parallelism of component labeling [Shi82], [Hum85], [Duff86], [Cyp89], [Emb89] on MIMD or fine grain SIMD architectures. But component labeling is just a step towards the final scene interpretation and the data structure efficiency for the next step must be taken into account for a global optimization of computer vision process.

We present in this paper, a component labeling algorithm developped in 1988 [Duc88b] for a coarse-grained parallel architecture named OPSILA. OPSILA is an experimental general purpose multiprocessor developped and built by the LASSY and supported by the DRET (research management of french army). OPSILA uses two different forms of parallelism like PASM [Sie81]. The first one is a well known SIMD mode (synchronous mode) and the second one is called SPMD (single Program Multiple Data stream) which is an asynchronous mode. Many applications have been implemented on OPSILA like finite elements [Shu87], ray tracing algorithm [For88] or particule accelerator [Mai88]. For the Computer Vision, we have shown OPSILA's effectiveness for the low-level image processing [Duc88a] and we demonstrate here its effectiveness for the intermediate level with a component labeling implementation. The aim of the implementation is obtained after processing the best load balancing efficiency. If we say the processor load depends on the number of region pixels, we want after processing the same number of pixels in each processor. In general, this involves that we won't get the same number of region in each processor and therefore involves a communication cost between processors which must be taken into account.

So, we propose a load distribution algorithm which incorporates a tolerance factor on the load function. This algorithm can take into consideration the initial localization of region and we show it is the better choice to the load balancing efficiency versus the cost of list transfer. Some examples and results are given on real image.

2 Presentation of OPSILA

OPSILA is a general purpose parallel architecture belonging to the mixed class (see [Aug87] for more details). It runs with two different forms of parallelism. The first one is the well known SIMD mode (a synchronous form of parallelism) [Ree81]. The second one is called SPMD (single Program, Multiple Data Stream) which is an asynchronous mode. These two modes are dynamically configurable in one machine cycle. OPSILA is composed of two parts : a central control unit and a parallel computation unit, which contains $p = 16$ processors, p memory units and an Omega/Benes interconnection network [Feng81]. In SIMD mode, memory is shared : each processor can access at any address of memory. In SPMD mode, each processor accesses only at its local memory.

The scalar-control unit performs overall management of the system. It consists in two processors : the Scalar Processor (SP) and the Instruction Processor (IP). The IP manages the vector unit (memory, network, vector instructions and synchronisations between SIMD and SPMD modes).

The vector unit consists of $p=2^k=16$ Elementary Processors (PEs), each associated with a memory bank (MB). A synchrone Omega/Benes interconnection network is used to perform data exchanges. Particular facilities are provided as any length vectors, automatic memory and network management, and indirect parallel addressing GATHER and SCATTER. OPSILA is built with bit slice technology. The processors are AMD 29116. Each MB capacity is 96 Kbytes. The base cycle time of the machine is 700 ns. Operands can be integers of 16 or 32 bits or 32 bits real numbers. (see figure 1 for OPSILA block diagram)

The SIMD operating mode :

In SIMD mode, the application program is entirely stored in the SM and managed by the SP. The data (vectors and matrix) are located in the vector memory. At the same time, each processors executes the same instruction on different data. Two consecutive addresses in vector memory are in adjacent memory banks. A linear vector is such that the addresses of successive components are in arithmetic progression. Each vector is described in SM by a pair (E, R). E is the address of the first element in the vector memory, and R is the common addresses difference for consecutive elements. In order to execute operations between vectors of any length, the IP segments at running time the vectors into subvectors of a length at most equal to the number p of EPs.

When the length of subvector is less than p, the EPs containing no significant data are masked. the IP handles any access conflicts (R is odd) to the vector memory. Indirect vector addressing instructions GATHER and SCATTER have been defined to extend domain applications of the SIMD mode [Shu87].

The SPMD operation mode :

Simultaneous and separate execution of several instructions flows is the most efficient way for handling applications or part of applications that are almost sequentially processed on purely SIMD or pipeline machines. In SPMD mode, the same program is duplicated in each MB. OPSILA is then composed of a set of 16 independant sequential processors, each associated with the same number memory bank. Under SPMD mode, PEs cannot exchange informations. Date exchanges can only occur in SIMD mode via the synchrone interconnection network.

SPMD mode is initialized by the IP which provides each PE the starting SPMD code address. The machine can be dynamically configured in SIMD or SPMD mode. This allows us to process vector or matrix globally in SIMD mode and locally in SPMD. The synchronizations required for initializing the SPMD mode and for returning to SIMD mode are a fork-join operating over the set of PEs. As the synchronization mecanism is simple, its operation is very efficient : transition from SPMD to SIMD mode is made in one machine cycle after the end of execution of the PE with the largest work load.

Programming and environment:

The implementation of parallel algorithm on OPSILA is carried out with a high level structured language named HELLENA [Jeg86], a PASCAL like, improved by :

- Vector instructions defined by extending saclar operations to array of scalars (by overloading of operators) and executed in SMID mode
- Block instructions used to enter and to leave the SPMD mode.

OPSILA is connected to a micro-vax and a high resolution display.

3 Parallel implementation of the component labeling

3.1 Sequential algorithm

The main idea of the algorithm is to assign different components with different labels [Bal82]. A component is defined as a set of pixels connected in one of the eight directions (8-connectivity). The labels are like equivalence tables with equivalence relation "belongs to the same component as". For our propose, the result we want obtain is a identified list for each equivalence table

The algorithm scans image, pixel by pixel, from left to right and from top to bottom, with a window defined in figure 2. For each current pixel, the algorithm tests the neighbourhood configuration and in function of this, makes some ones of the following operations:

- create a new label (the generated labels start from 1) and a new list.
- labeling the current pixel and put the pixel in a list
- if two neighbourhood pixels have different labels L_1 and L_2 with $L_1 < L_2$, write that L_2 is an equivalent table to L_1

After the scanning, the equivalence tables are reorganised and the lists of same equivalence are merging. So, with this algorithm, we make a component labeling in single pass over image.

The main difficulty of this algorithm is the pixel treatment is context-dependant. The process is not homogenous and not regular with a machine cycle. So, we can think that an asynchronous architecture fits better than a synchronous architecture. In the next section, we show an implementation with OPSILA's asynchronous mode.

3.2 Parallel implementation

For a coarse parallel architecture, component labeling sets following problems :

- Representation and treatment of dynamic structures (lists) in an mutiprocessor architecture. An objet parallelism is an asynchronous mode. Then we want a local management of lists
- List exchanges between processors needs a definition of communication tools to achieve objet parallelism.
- Load balancing efficiency for next treatments. The load balancing is a global process because it implies the knowledge of every processor information

In fact, these problems are general problems in intermediate and high level vision, and we illustrate them on a particular case. We propose the following algorithm

1. Divide the picture into 16 vertical bands with one column overlapping (SIMD mode).
   In fact, dividing image in regular square minimizes the length of frontiers, but it increases the communication costs with special treatments for corners which belong at 4 processors.
2. Run the sequential algorithm into every processor (SPMD mode). The algorithm runs in single pass.
3. For each processor, make a label merging process by exchange of overlapped vertical columns(SIMD and SPMD mode)
4. Compute a global labeling by a merging of 16 equivalence table and compute the load distribution between processors to obtain a homogenous load balancing (sequential mode realized by scalar processor)
5. Each processor puts sub-region list in a output mailbox for other processor (SPMD mode) in relation to the equivalence table. Mailboxes are exchange in SIMD mode. Then each processor merges sub-region lists which are in its input mailbox (sending from the other processors) (SPMD mode).

3.2.1 Optimal algorithm for load balancing

A load balancing is an application f : O $\longrightarrow$ P, where O is a set of n objects $O_1, ...O_n$ and P is a set of p processors $P_1, ..., P_p$ We define :

- m_i as the cost of object i for $i \in [1, n]$. In our case, the object is a region-list, and we consider the cost as the number of pixels.
- L_u as the cost of processor u, $u \in [1, p]$, where $L_u = \sum_i m_i$, i.e. equal the sum of object costs into processor u.

Now the problem is to minimize a global cost from elementary cost associated to object. A solution consists to find p^n solutions and to take the best.

So to avoid this, we introduce a quality criterius of the load distribution process which measures the efficiency Ec of load distribution

$$\text{Ec} = \text{Lt} \ / \ \text{p.Lmax}$$

where Lt = total load = $\sum_{u=1}^{p} P_u$
Lmax = load of the processor which is the most loaded
p = number of processor.
For Opsila, the best load function (Ec=1) is for

$$Lmax = L_i = L_j = Lt/16, i \neq j, 1 \leq i, j \leq 16 \qquad (1)$$

So, if q is the number of objects and p the number of processors, the optimal algorithm is :

1. Find the processor P_u the least loaded among p
2. Find the object O_i with the highest cost among the objets remaining
3. Put O_i in P_u
4. goto 1 until object list is empty

The algorithm complexity is $O(q^2 + pq)$. We can reduce this complexity by sorting objects before processing in function of object cost.

But this algorithm does not take into account the initial localization of objects versus processors. So, this algorithm does not minimize data transferts between processors as shown in figure 3.

3.3 Best efficiency with taking account the list transfer cost

To solve the problemm shown in figure 3, a second criterius must be found which takes into account costs of list transfers between processors to obtain a complete region in a processor. For this, we must limited the transfert.

So, we propose a load distribution algorithm which incorporates a tolerance factor on the load function. This algorithm can take into consideration the initial localization of region and it is the best choice to the load balancing efficiency versus the cost of list transfer. So, we introduce a notion of tolerance on Ec expressed in percent. This tolerance gives a set of the least loaded processors and we can choose among this, the processor for which the transfert cost is minimal. The optimal algorithm with load tolerance is :

1. Find the set S of the least loaded processors among p processors with a load tolerance t
2. Find the object O_i with the highest cost among the objets remaining
3. Give O_i to processor P, $P \in S$, if the choice P minimizes exchange cost.
4. Goto 1 until object list is empty.

4 Results

In this section, we give two kinds of results about component labeling implementation, firstly an implementation on OPSILA with the first algorithm of load balacing. Secondly, we present a simulation to illustrate the behavior of modified algorithm of load balacing.

4.1 hardware constraint

OPSILA is a prototype of a parallel architecture. Because each memory bank holds only 49kb of memory, we limit the size of input image to be less than 256x256 pixels. In addition, memory saturates with 200 lists per processor (representing about 800 pixels). So, we have chosen to realize a component labeling on a edge map of a image. On OPSILA, we have computed an edge detection with non maxima suppression and taken 20 per cent of these edges. Initial image is the well known image of girl Lena. For this image, we have the following characteristics (with 16 processors):

- Number of edge-pixels is 6273 representing 396 different lists
- Average length of lists : 15.8
  Average number of lists per processor 24
- Average cost per processor : 392

4.2 Real implementation

We shown of figure 4 a complete analysis of the load balacing, using the first algorithm. We show in figure 4.a processor load (i.e. number of pixel) before and after sharing out. The efficiency after processing is $0.76 = 6273/16 * 510$, where 510 is the load of processor No 13. We show in figure 4.b list distribution after labeling (first step), and after global distribution in relation with figure 4.a. The number of list is reduced because of list merging. Figure 4.c presents results about emitted and received pixels for each processor. We have found that 3935 pixels (62 per cent of pixels) representing 173 lists have been moved.

We present on figure 5 run time in function of number of processors we have activated. Time is given in Megacycle of OPSILA (cycle = 700 ns). Component labeling time decreases as we use more processors. Then, with 16 processors, we obtain about 2 secondes. With 15 processors, we have a bad localization of pixels, i.e a non homogenous distribution before labeling. The 14 processors must wait the 15th which is the most loaded processor during labeling. This effect can be reduce by an adaptive column cut.

4.3 Comparison between both algorithms of load balancing

We present in figure 6 a result of a simulation between the optimal algorithm of load balancing and the algorithm with tolerance of load. We show the load balancing efficency is constant and the cost of transfer decreases in function of tolerance parameter.

5 Conclusion

We have demonstrated in this paper the effectiveness of general purpose parallel architecture OPSILA for the intermediate level with a component labeling implementation. So, we have proposed two algorithms of load distribution and shown the benefit to introduce a tolerance factor on the load function for a better load balancing efficiency. We have made an implementation and tested the algorithms on real image.

References

[Aug87] M. Auguin : "Experiments on a parallel SIMD/SPMD architecture and its programming", in France-Japan Intelligence and Computer Science Symposium, Nov 1987

[Bal82] D.H. BALLARD et C.M. BROWN : "Computer Vision" , Prentice Hall, New Jersey (1982).

[Cyp89] R. Cypher, J.L.C. Sanz, L. Snyder : "An Eprew Pram Algorithm for Image Component Labeling", in PAMI, Vol 11 , March 1989, pp 258-261

[Duc88a] P. Duclos, F. Boeri, M. Auguin, G. Giraudon : "Image Processing on a SIMD/SPMD Architecture : OPSILA", in Proc IEEE of 9th ICPR, Roma, November 1988, pp 430-433.

[Duc88b] P. Duclos : " Etude du parallelisme en traitement d'images. Realisations sur architecture mixte SIMD/SPMD", PhD, Universite of Nice 1988 (In French)

[Duff86] M.J.B. Duff ed., Chapter 7 in Intermediate-Level Image Processing, Ac. Press London 1986

[Emb89] H. Embrechts, D. Roose, P. Wambacq : "Component labeling on a distributed memory multiprocessors", in Proc of the first Europeam workshop on hypercubes and distributed computers, Rennes 1989, North-Holland, pp 5-17.

[For88] M.C. Forgue: Parallelisation du lancer de rayons sur un calculateur de type SIMD/SPMD, PhD of Universite of Nice, Septembre 1988 (in french)

[Feng81] T.Y. Feng : "A Survey of Interconnection Networks", in Computer pp 12-27, December 1981.

[Jeg86] Y. Jegou : "Le langage vectoriel Hellena", Rapport de Recherche INRIA No 703, July 1986 (in french)

[Hum85] R. Hummel, A. Rojer: "Implementing a Parallel Connected Component Algorithm on MIMD Architectures", IEEE Workshop on Computer Architecture for Pattern Analysis an Image Data Base Management, Miami Florida 1985

[Han78] A.R. HANSON et E.M. RISEMAN : "VISIONS: A Computer System for Interpreting Scenes", New York Academic Press pp 303-333 1978

[Mai88] J. Maillard, J. Siva, M. Auguin , F Boeri : "Accelerator Simulation on a Parallel Computer", European Conference on Accelerator, Rome June 1988.

[Ree81] A.P. Reeves : "Parallel Computer Architecture for Image Processing", pp 199-206, ICCP 1981

[Shi82] Y. Shiloach and U. Vishkin: "An O(log n) Parallel Connectivity Algorithm", in Journal of Algorithms, 3, pp 57-67 (1982)

[Shu87] K. Schubert : "Vectorisation de la methode des elements finis sur un calculateur de type SIMD/SPMD", PhD University of Nice, Octobre 1987, (in french)

[Sie81] H.J. Siegel andP.T. Kermerer : "PASM : a partitionable SIMD/MIND System for image processing and pattern recognition", IEEE Trans on Computers, Vol c30 No 12. December 1981.

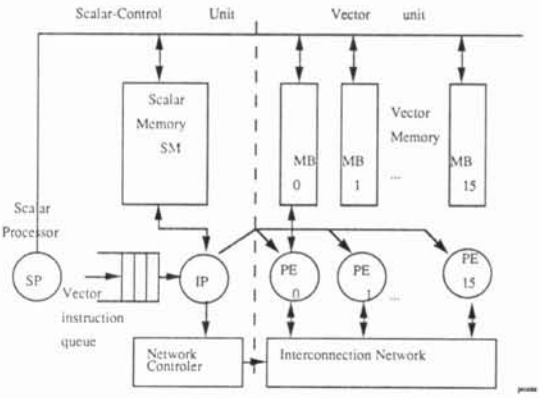

Fig. 1 OPSILA block diagram

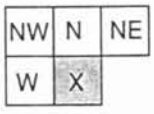

Figure 2: Mask used in the algorithm

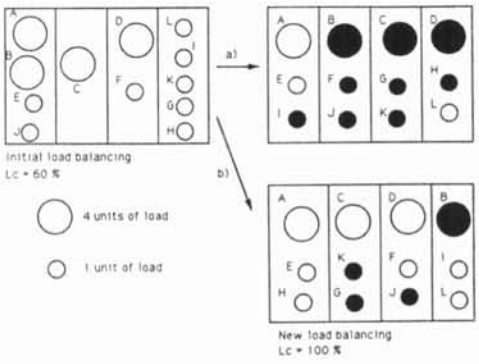

Figure 3 Two examples of load balancing giving same efficiency Lc. The black objects are moved
a) optimal algorithm b) Modified algorithm

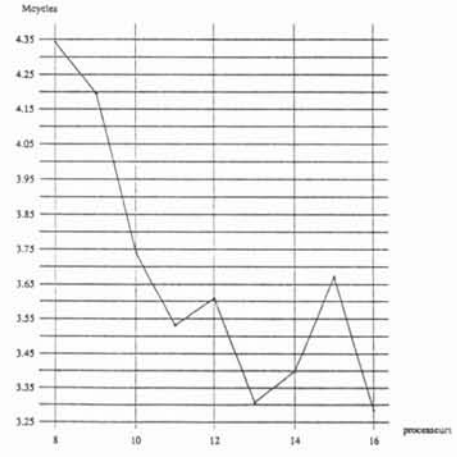

Figure n° 5 : Run time versus number of processors

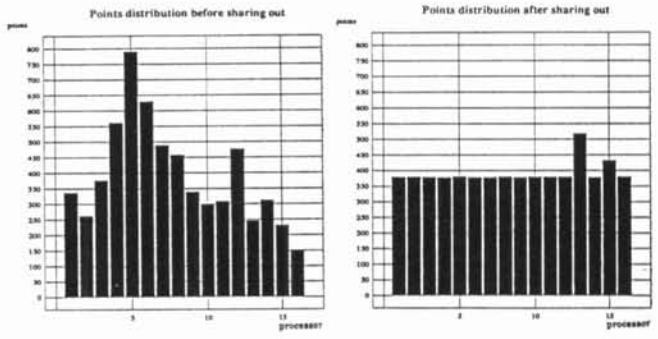

Figure n° 4 a

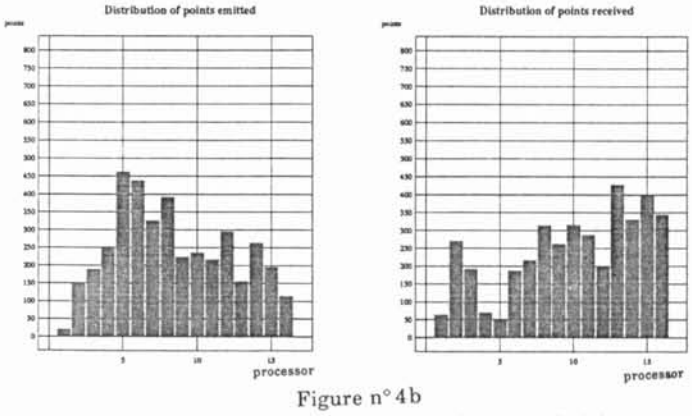

Figure n° 4 b

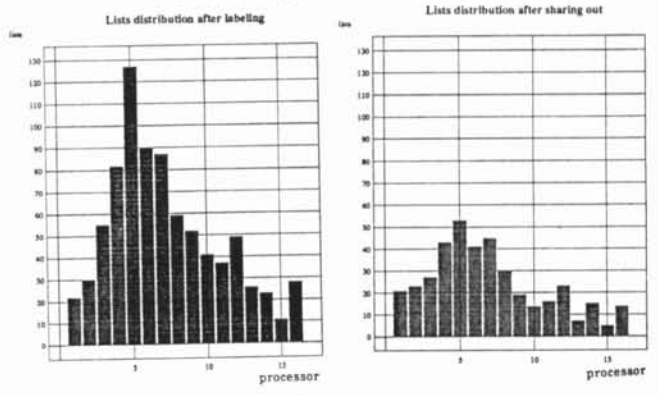

Figure n° 4 c

Figure n° 4

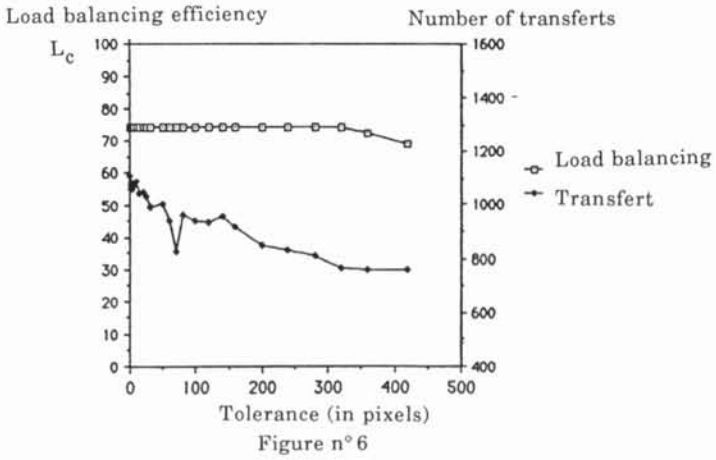

Figure n° 6