Morphological Hough Transform on the Instruction Systolic Array

Bertil Schmidt[1], Manfred Schimmler[2] and Heiko Schröder[3]

[1] ISATEC GmbH, D-24118 Kiel, Germany
[2] Braunschweig University of Technology, Germany
[3] Loughborough University of Technology, UK

Abstract. Instruction systolic arrays have been developed in 1987 [La1] in order to combine the speed and the simplicity of systolic arrays with the flexibility of MIMD parallel computer systems. In this paper a new algorithm for line detection is presented which applies the morphological approach to the well known Hough transform. The quality of its results is significantly higher than that of the classical Hough transform. This new algorithm has been tailored towards the capabilities of the instruction systolic array. It has been implemented on the *Systola 1024,* the first parallel computer of this particular architecture on the market. Systola 1024 is an low cost add-on board for standard PC's with PCI slots.

1 Introduction

Instruction systolic arrays (ISAs) provide a programmable high performance hardware for specific computation intensive applications [DS1,Sch1]. Typically, such an array is connected to a sequential host, thus operating like a coprocessor which solves only the computationally intensive tasks within a global application. The ISA model is a mesh connected processor grid, where the processors are controlled by three streams of control information: instructions, row-selectors and column-selectors. The concept of instruction systolic arrays is explained in detail in the second section.

Hough transform is a standard technique for line detection in image processing. It is based on the accumulation of information about straight lines intersecting with points (pixels) of the image plane. The details of Hough transform are given in section 3. The classical Hough transform creates artefacts if there are structures in the image like dashed lines or small line segments. Therefore, modifications of Hough transform have been developed [BPS1,Lea1]. In this paper an approach based on mathematical morphology is taken to find an efficient implementation of Hough transform that minimises the number of artefacts in the transform space. This algorithm is explained in section 4.

The implementation of this new algorithm is presented in section 6. For this purpose the Systola 1024 is shortly explained in section 5. It is the instruction systolic computer architecture on which the algorithm has been implemented.

Section 7 discusses the performance in comparison to implementations on a sequential architecture. The results of the original Hough transform and those of the new algorithm are demonstrated in an example.

2 Principle of the ISA

The ISA is a quadratic array of identical processors, each connected to its four direct neighbours by data wires. The array is synchronised by a global clock. The processors are controlled by instructions, row- and column-selectors.

The instructions are input in the upper left corner of the processor array, and from there they move step by step in horizontal and vertical direction through the array. This guarantees that within each diagonal of the array the same instruction is active during a single clock cycle. Processor *(i+1,j)* and *(i,j+1)* execute in clock cycle *k+1* an instruction that has been executed by processor *(i,j)* in clock cycle *k*.

The selectors also move systotically through the array: the row-selectors horizontally from left to right, the column-selectors vertically from top to bottom (Fig. 1). The selectors mask the execution of the instructions within the processors, i.e. an instruction is executed if and only if both selector bits, currently in that processor, are equal to one.

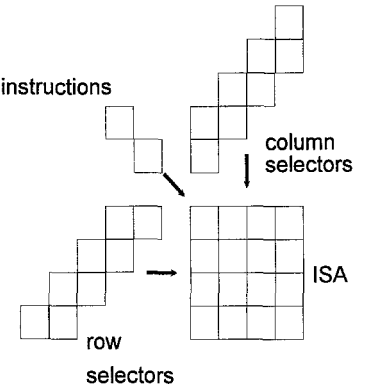

Fig. 1: Control flow in an ISA

Every processor has read and write access to its own memory. Beside that, it has a designated *communication register (C-register)* that can also be read by the four neighbour processors. Two adjacent processors can exchange data by writing on their own C-register and afterwards reading the C-register of the other in two subsequent clock phases. Within one clock phase the reading access is always performed before the writing access. This convention on the one hand avoids read/write-conflicts, on the other hand it creates the possibility to broadcast information across a whole processor-row or -column with one single instruction. This property can be exploited for an efficient calculation of row-sums and row-ringshifts which are the key-operations in the parallel Hough transform implementation described in section 6:

Row-sum. One important advantage of ISAs is the capability of performing aggregate functions within one (or a constant number of) instructions. Aggregate functions are operations where every processor needs information of all processors with smaller column index and the same row index (or vice versa). The simplest example is the computation of the row sums: Each processor computes the sum of the C-registers of its left neighbour and itself. Since the execution of this operation is pipelined along

the row each processor accumulates the sum of all C-registers up to its own which is identical to the prefix sum.

Row-ringshift: The contents of the C-registers can be ringshifted along the processor rows by two instructions. Every two horizontally adjacent processors exchange data (using one read left and one read right operation). Because of the instruction flow from west to east this implements a ringshift. Of cause, a column-ringshift can be executed in the same way.

3 Hough Transform

The classical Hough transform (HT) [IK1] is a very powerful technique to detect straight lines in a binary image. It transforms the image into a parameter space by counting pixels on straight lines. For every possible straight line in the image a counter is introduced which accumulates the number of pixels which satisfy the line equation $y = mx + c$. In the parameter space we represent a straight line by these two parameters: the slope m and the intercept (the intersection point with the y-axis) c. This representation is rather simple. The set of straight lines intersecting in one point of the original image is represented by a straight line in the m-c-space. The disadvantage of this representation is the fact that vertical lines (with infinite slope) cannot be represented. However, as shown in section 6, this does not cause problems in our implementation.

The Hough transform algorithm proceeds now in the following way: An accumulator array $B[m_k,c_l]$ is introduced to represent the parameter space. For every white pixel in the image and every slope m_k the corresponding value c_l is computed from the line equation and the counters B for all straight lines intersecting in this pixel are incremented by one. By that at the end of the execution the value of each counter represents the fraction of the corresponding straight line in the original image. The local maxima of the B array indicate the presence of lines of certain slopes and intersects in the image.

The HT algorithm in the simplest form for a binary image $I(i,j)$ of size NxN and an accumulator array $B[m_k,c_l]$, where $k = 0,\ldots,M-1$, $m_k = k/M$, looks like this:

Alg. 1: Standard HT

```
for i:=0 to N-1 do
  for j:=0 to N-1 do
    if I(i,j)=1 then
      for k:=0 to M-1 do increment B[m_k,round(j-i·k/M)]
```

One disadvantage of HT is the high requirement in computing power such that on-line computations are impossible on existing sequential computers. The work is of the order N^2M.

Therefore, a parallel implementation is useful and necessary for many applications. Another disadvantage is the fact that HT creates artefacts by counting structures in the image that finally turn out to be only very small fractions of straight lines (e.g. only one dot). In [BPS1] a new method to solve this problem is presented. Its basic idea and the new algorithm derived from it is presented in the next section.

4 Morphological Hough Transform

From the point of mathematical morphology the identification of a white pixel is the erosion operation with a structural element that consists of only one dot. Whenever the Hough transform algorithms finds a white pixel, it increments the counter for every straight line containing this pixel. By using a structural element which represents a straight line by more than one dot we can significantly reduce the number of counter increments. In particular, all those cases where the white pixel was isolated or consists of only a small white segment do not lead to an increment of any counter. This suppresses the artefacts in the *m-c*-space. Ideally, a straight line can be represented by two pixels (because there is exactly one straight line intersecting with these two pixels). By using the morphological approach with a structural element consisting of two pixels for every possible slope we increase a counter if and only if both pixels match in the original image.

Alg. 2 shows an implementation of this method (with $I(i,j)$, $B[m_k,c_l]$ like in Alg. 1 and the structuring element for each slope k has the distance (q_k,p_k) from the actual pixel-coordinates). In order to produce a parallel implementation on the ISA a much more efficient version of the morphological approach becomes possible if the loops in this program are swapped as in Alg. 3.

Alg. 2: MHT

```
for i:=0 to N-1 do
  for j:=0 to N-1 do
    if I(i,j)=1 then
      for k:=0 to M-1 do
        if I(i+q_k,j+q_k)=1 then
          increment B[m_k,round(j-i·p_k/q_k)]
```

Alg. 3: MHT with swapped loops

```
for k:=0 to M-1 do
  for i:=0 to N-1 do
    for j:=0 to N-1 do
      if I(i,j)=1 and I(i+q_k,j+p_k)=1 then
        increment B[m_k,round(j-i·p_k/q_k)]
```

Now we perform the accumulation operations for the complete image slope by slope. Every slope is represented by a structural element of two pixels, e.g. *(0,0)* and *(x,y)*. By shifting the image by *(-x,-y)* we get two copies of the image on top of each other. Now the morphological erosion can be performed by one AND-operation of the original image with the shifted image.

In reality, there may occur uncertainties due to rounding effects: Not every straight line has a slope which can be exactly represented by two pixels with a constant distance. The solution of [BPS1] selects a collection of structural elements that are almost equidistant. Skewing the image and choosing an appropriate structural element can avoid this problem. The details of this solution as well as the parallel implementation of the morphological erosion with the shifted image are given in the section 6.

5 Architecture of Systola 1024

The parallel computer Systola 1024 is an low cost add-on board for standard PCs [LMS1], providing a *32x32*-ISA. It consists of a *4x4* array of processor chips. Each chip contains *64* Processors, arranged as an *8x8* square.

In order to exploit the computation capabilities of this unit, it is necessary to provide data and control information at an extremely high speed. Therefore, a cascaded memory concept is implemented on board that forms a fast input and output environment for the parallel processing unit (Fig. 2).

For the fast data exchange with the processor array there are rows of intelligent memory units at the northern and western borders of the array. These units are called *interface processors*. Eight interface processors are integrated onto one chip. Each interface processor is connected to its adjacent array processor by a single wire for data transfer in each direction.

The interface processors have access to an on board memory by means of special fast data channels, those at the northern interface chips with the northern board RAM, and those of the western chips with the western board RAM. The board RAM can communicate bidirectionally with the PC memory. The data transfer between every two memory units within this hierarchy is controlled by an on board controller chip. In particular, it controls the channels between the interface processors and the board RAMs as well as between the board RAMs and the PCI bus of the PC.

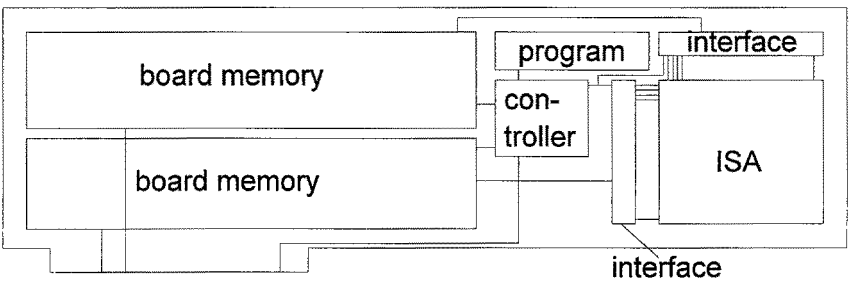

Fig. 2: Block diagram of the Systola1024-board

In addition, the controller supplies the interface processors and the array processors with instructions and selectors that are stored in an additional memory on board, the *ISA program memory.* The ISA program memory can contain a large variety of programs for the processor array, according to the different tasks that may be necessary to perform during one application. Every word clock cycle the controller transmits one instruction, one row-selector and one column-selector from the ISA program memory to the interface processors. The instruction consists of *32* bits as well as the row-selector and the column-selector. A triple of instruction, row-selector and column-selector is called a program diagonal. The ISA program memory can contain *16 K* program diagonals.

The controller receives its instructions either directly from the PC as so *board instructions*, or it can operate autonomously. In the second case it receives *controller instructions* from an *instruction queue,* which is located on the board, too. The instruction queue can be loaded from the PC and it consists of up to *256* controller instructions, each of them in *16* bit format.

Systola 1024 has a peak performance of *3200* MIPS. In practice the performance lies, dependent on the application, between *1000* and *3000* MIPS.

6 Parallel Implementation of Morphological Hough Transform

For the implementation of the MHT we take a *NxN, N=512,* binary image. Every processor of *Systola 1024* stores a *16x16* subimage in *16* internal registers. So each of this registers stores *16* adjacent pixels of the same row. The capabilities of ISAs to compute row (column)-sums and -ringshifts very efficient (see section 2) are exploited in the parallel implementation:

The erosion and accumulation for horizontal (vertical) lines can be done very fast: For a structuring element of two pixels with distance *d*, we can shift a copy of the image above the origin by *d* pixels. Now a counter has to be incremented if a pixel of both, the origin and the shifted image are set. With shearing of the image other line angles are transformed in horizontal position, such that the efficient horizontal operations are always applied.

Horizontal Accumulation. This operation means to sum up the pixels of each image-row. The implementation first builds the subrow-sum within each processor in its C-register. Afterwards the efficient processor-row-sum computes the whole image-row-sum. This operation is repeated *16* times for every subimage-row of each processor. At the end of the iteration the last processor-column stores the results. Because of the limited memory (*32* registers) of each processor the results are shifted one processor-column to west in every second iteration step. The *i.th* processor-column always collects accumulator-entries for the line-intercept *i*. When all processor-columns are covered the accumulator-array-results are transferred to the board-RAM.

Horizontal Erosion. Let the structuring element have the distance *d* from the image-origin. Then an image-row-ringshift for *d* positions and an AND-Operation computes the horizontal erosion. In Systola an image-subrow is ringshifted one processor. Afterwards a shifting for *(d mod 16)* positions within each processor is performed. Then a ringshift for a *(d div 16)*-processor-distance computes the image-row-ringshift. Finally one AND-Operation between the origin sub-row-register and the ringshifted sub-row-register in each processor calculates the erosion. After that the result is vertical accumulated.

Shearing. The shearing for line-angles between $0°$ and $45°$ is an image-column-ringshift, where each column has a different shifting-distance depending on the line-angle. To avoid complex operations this distances are precalculated.

For the implementation we take *N* slopes between $0°$ and $45°$. So the following line equations are considered: $y = (k/(N-1))x + c$; $k=0...N-1$; $c=0...N-1$. The different line-angles are handled in increasing order. Then the shear operation only needs to incorporate the relative shear between neighbouring angles. So the image-column-ringshift only has to consider the next column. The precalculated shearing-distances turns to a binary image-row-mask: Only the image-columns with an entry in the corresponding mask-position has to be ringshifted.

The angles between $0°$ and $-45°$ are computed by loading the input image with transposed rows on the ISA and applying the same program. The angles between $45°$ and $90°$ ($-45°$ and $-90°$) are calculated by storing the *16x16*-subimage column-wise in

each processor and applying the reflected ISA-program (swapping row- and column-selectors, changing west and north, east and south) for $0°$ to $45°$ ($0°$ and $-45°$). The finally accumulator size is $4N$ (slopes) x N (intercepts).

7 Performance and Experimental Results

To compare the performance of the parallel algorithm in section 6, we compare the implementation on Systola 1024 with two sequential versions on a PC (Pentium Processor, 200 MHz, optimised code). The first version is the standard version of the HT-algorithm (referred to as *"P200 Standard"* in Table 1). The second version is a sequential simulation of the algorithm implemented on Systola 1024 (referred to as *"P200 Parallel"* in Table 1). For a comparison to other parallel architectures for the Hough transform see [Fer1].

For the implementations we use binary images of size NxN, $N=512$, and an *8-bit* accumulator array consisting of $4N$ slopes and N intercepts. Systola needs *0,25 s* (*375* processor instructions per slope) for the standard HT and *0,31 s* (*471* instructions per slope) for the MHT. Since computing time dominates communication time in this application, data transfers can be almost totally hidden as it can be executed concurrently to the computation.

The standard HT (Alg. 1) depends on the number of white pixels in the input image, but has the disadvantage of a complex computation of the accumulator bins to be incremented. This means a computing time of *70,0 s* in the worst case (all pixels white). The used edge images from the pick-and-place process of multichip-modules (Fig. 7) have *10%* seeded pixels on an average, which corresponds to a computing time of *7,0 s*.

The standard MHT (Alg. 2) depends on the number of seeded pixels in the input image and in each eroded image. An average multichip-module edge image (Fig. 7) has *10%* seeded pixels in the input and *1%* in each eroded image, which leads to a computing time of *5,7 s* in the average case.

The sequential version of the parallel algorithm in section 6 is independent of the number of white pixels, but has the advantage of an easier accumulation. This leads to a constant computing time of *5,0 s* for the HT and *6,2 s* for the MHT for each image.

Table 1. Comparisons of the average performance of HT/MHT for a *512x512* image and *2048* different slopes on Systola 1024 (including data transfers) and Pentium 200

	Systola 1024	P200 Standard	P200 Parallel	Speedup
HT	0,25 s	7,0 s	5,0 s	28 / 20
MHT	0,31 s	5,7 s	6,2 s	18 / 20

In order to evaluate the qualitative performance of the MHT we have compared it with the HT on several different images.

Fig. 3 shows three lines close together. The relevant part of the accumulator array from the HT and MHT (with distance $d=50$ for the structuring element) is displayed as a height field in Fig. 4 and 5. It can be seen that both, HT and MHT produce artefacts (in the Figures they are behind the three peaks representing the lines). In case

of MHT these artefacts are significantly reduced. The MHT also creates an enhanced accumulator contrast, which simplifies the higher order image analysis.

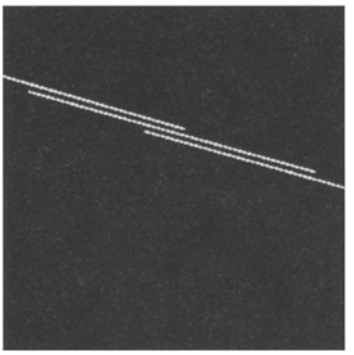

Fig. 3: Original Image

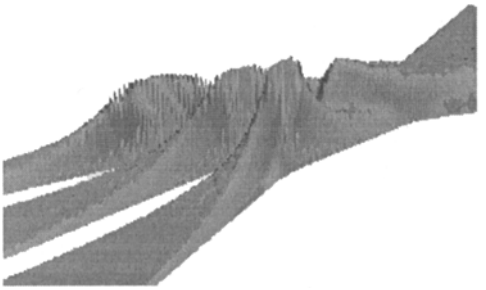

Fig. 4: Accumulator from HT

Fig. 5: Accumulator from MHT

Comparisons of the effectiveness of the HT and MHT methods are displayed in Fig. 6-9. Fig. 6 shows an image from the pick-and-place process of multichip-modules. Fig. 7 shows the result after an edge detector is performed on the image. Fig. 8 and 9 show the result after a HT and MHT (*d=10*), where accumulator entries greater than *90* and *80* are detected. The resulting lines are displayed after an AND-operation with the edge image. It can be readily seen that the HT picks up many more false lines than the MHT in the dotted areas.

Obviously there is still lots of scope for fine tuning the combination of HT and morphology to best fit a given application. For example, the structuring elements could be optimised to search for lines of a given thickness (using larger structuring elements), of a given length or which are dotted. We attempt to work with this method further in the area of automatic quality control and image classification [Ko1].

References

1. Beresford-Smith, B., Pham, B., Schröder, H.: A parallel morphological implementation of the Hough transform, Proc. 25th. HICSS, 111-119 (1992).
2. Dittrich, A., Schmeck, H.: Given's Rotation on the Instruction Systolic Array. In: Wolf, G., Legendi, T., Schendel, U. (eds.): PARCELLA '88, Mathematical Research, Bd. 48, Akademie Verlag, Berlin, 340-346 (1988)

3. Ferretti, M., Albanesi, M.G.: Architectures for the Hough transform, Computer Vision, Graphics and Image Processing 44 (1996) 542-551
4. Illingworth, J., Kittler, J.: A survey of the Hough transform. Computer Vision, Graphics and Image Processing, 44, 87-116 (1988)
5. Kolbe, W.: Online Qualitätskontrolle von Oberflächen mit den ISATEC Surface Quality Scannern SQS, Journal für Oberflächentechnologie, 3 (1997)
6. Lang, H.-W., The Instruction Systolic Array, a parallel architecture for VLSI, integration, the VLSI Journal 4, 65-74, (1986)
7. Lang, H.-W., Maaß, R., Schimmler, M.: The Instruction Systolic Array - implementation of a low cost parallel architecture as add-on board for Personal Computers, Proc. HPCN 94, Munich (1994)
8. Leavers, V.F.: Which Hough transform, Computer Vision, Graphics and Image Processing 58 (1993) 250-265
9. Schimmler, M.: Fast sorting on the Instruction Systolic Array, Technical Report No. 8709, Christian-Albrechts-University, Kiel, (1987)

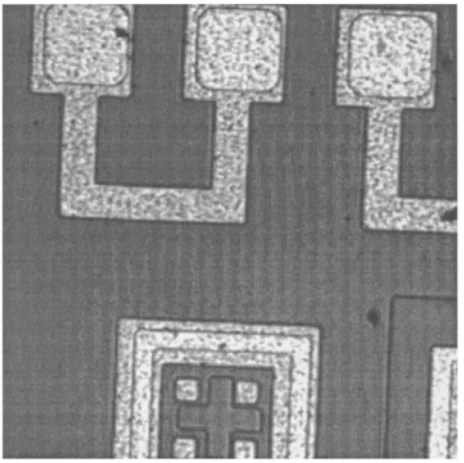

Fig. 6: Original image

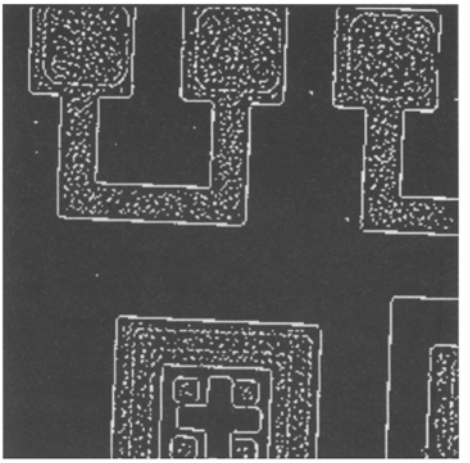

Fig. 7: Edge image

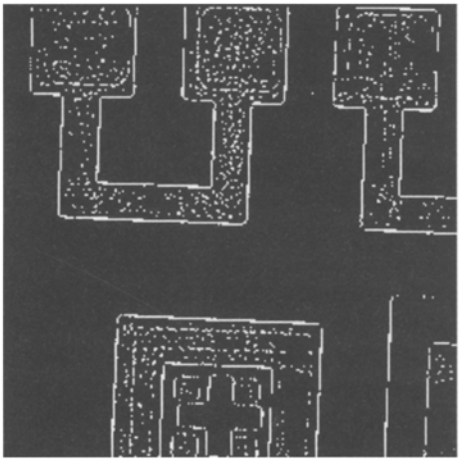

Fig. 8: Result from a HT

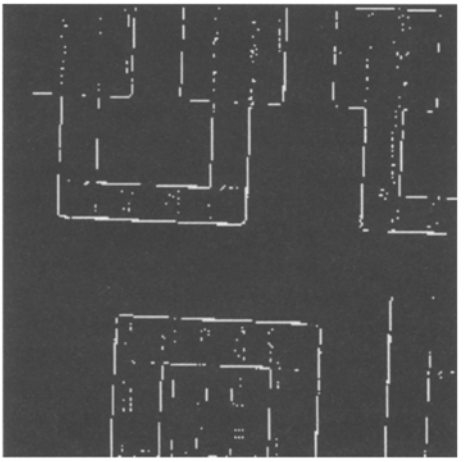

Fig. 9: Result from a MHT