Beitrag zum 17. DAGM-Symposium Mustererkennung 1995.
13.-15. September 1995, Bielefeld

A VLSI system for linear and non-linear local image filtering*

Bernhard Lang and Manfred Troike
MAZ Hamburg GmbH, Harburger Schloßstraße 6-12,
21079 Hamburg, Germany
+49/40/76629-2001, email: lang@maz-hh.de
+49/40/76629-1441, email: tk@maz-hh.de

Abstract

This paper presents the GIPSi processor array which is a programmable VLSI chip designed for local linear and non-linear image filtering algorithms. A first section introduces the GIPSi approach. Then the parallel SIMD architecture is described followed by a section about implemented algorithms. Special emphasis is laid on a rank order filter algorithm which has been adapted to the SIMD architecture. It follows a section concerning the VLSI implementation of the GIPSi system and some conclusions.

1 Introduction

The GIPSi[1] architecture is designed for linear and non-linear image preprocessing algorithms. Examples are linear algorithms like low-pass filtering and edge-detection and nonlinear algorithms like erosion or median filtering. More general speaking the GIPSi system can handle mappings from a source image I_s to a result image I_r where a line sequential data representation is assumed for both, source and result images. Further a pixel $r_{x,y} \in I_r$ of the result image should only depend on a local window of the source image which surrounds the corresponding source pixel $s_{x,y} \in I_s$.

Due to the easy and regular kernels it is possible to build adequate hardware for classes of image preprocessing algorithms. Different architectures have been proposed [Ste83, Ree84, Kun88]. One approach describes a *parallel line scan architecture* which is formed by a one-dimensional SIMD array of simple processors [FH85] and related data input and output shift registers. It shows a very high flexibility because it allows the implementation of algorithms on a software basis.

2 Line scan SIMD architectures for image processing

Figure 1 shows an overview of a system using the addressed SIMD approach. The 1-D array includes 3 main execution units which work in parallel: an input shift register, an array of processors P (each with private memory M) where computing occurs, and an output shift register. A further unit described as *sequencer* supplies a common instruction stream to all processors of the system. One processor is available for each image column, thus one result line can be computed in parallel by a SIMD program. When a new line of the source image is clocked into and a previously computed result line is clocked out of the system a new result line can be computed in parallel. Thus computing can be hidden behind the incoming and outgoing data streams.

*This work has been done in cooperation with TU Hamburg Harburg, Technische Informatik I.
[1]GIPSi: General Purpose Image Preprocessor in Silicon

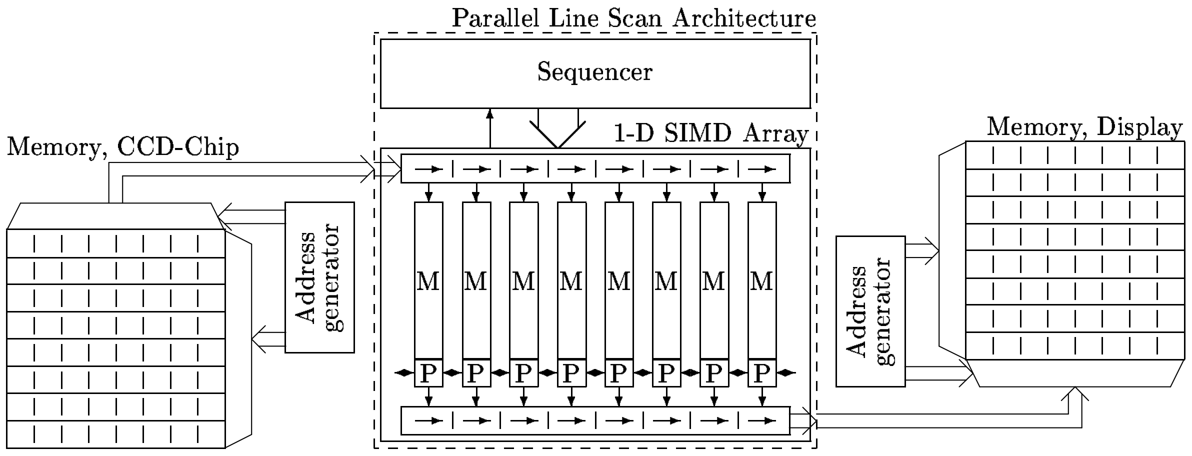

Figure 1: SIMD Processor Line for image processing tasks

Existing Systems: Some systems have been designed using the described approach. A first system named AIS-5000 [Wil88] shows a very low integration level. Only 8 bit-serial processors without memory fit on one chip. The memory has to be supplied by further chips.

A very high integrated approach has been presented in [MKW$^+$90] where a serial video processor (SVP) is introduced for digital TV. It consists of up to 1024 processors on one chip each equipped with two memory banks of 128 bits. An instruction generator (sequencer) is included on chip. The chip would be suitable for image processing demands but due to its consumer market orientation it is only available in very large lots. This prevents or even limits its use as an image processing accelerator for industrial applications.

Another work [CS91] describes a video/image processor (VIP) where 512 processors each including a memory of 128 Bits are integrated on one chip. Its processing units are specially designed for image processing algorithms. However the chip forms only the core of a desired accelerator and must be extended by an instruction sequencer. Further as a research product it is not available.

A current system [YKF94] integrates 64 8-Bit processors with 32 blocks of 64 *kBit* memory on one chip. Yet its current status is *"experimental product"* and no intention exists to produce it.

The GIPSi system: GIPSi fills the described gap. It includes a high integrated core of 512 single bit processors on a single chip. Input and output of image data is handled by a flexible protocol [Lan94a]. This protocol allows an easy adaption of surrounding hardware. Further a sequencer chip is planned which generates instruction streams by expanding a very dense program representation. Programming of GIPSi is supported by a symbolic macro assembler. Program debugging can either be done at the real hardware or using a simulator program. The simulator program allows a very transparent look into the system and eases the evaluation of new programs.

3 GIPSi programming model

The programmer can look to the GIPSi system from two different views. The first view shows a global overview, the whole processor array is regarded as one block surrounded by the supporting units. This view is important for setting up the system via a controlling host processor and for understanding the global system behavior. The second view looks to one single processor of the SIMD array and shows the local processor architecture. This view is important for writing image processing programs. A GIPSi assembler language has been developed to support programming at that level.

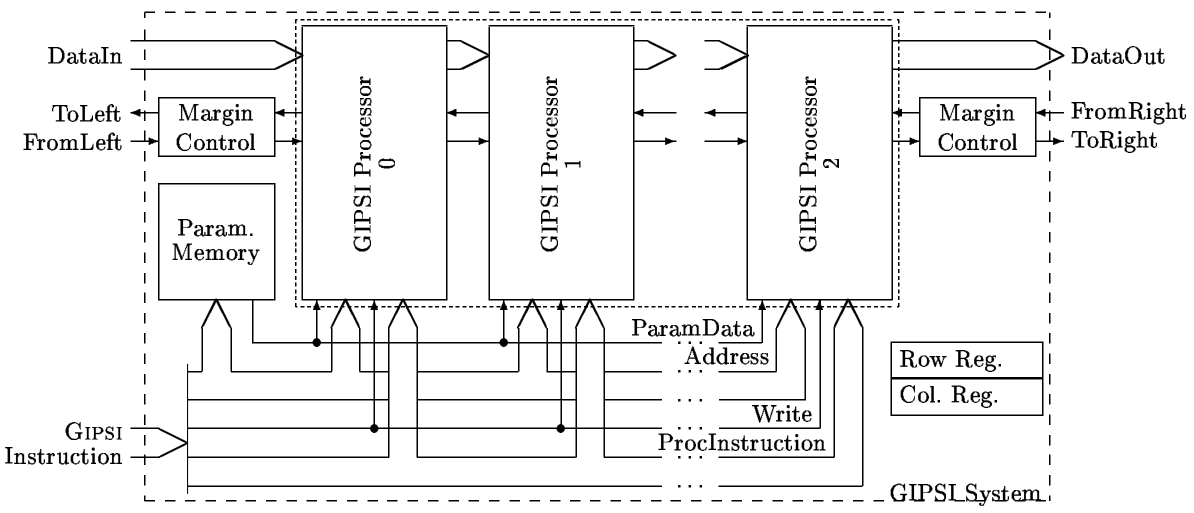

Figure 2: Programmers global view to the GIPSi system

3.1 Global view to the GIPSi system

Globally the GIPSi system consists of a linear array of P single bit processors and some further blocks. Their arrangement is shown in Figure 2.

GIPSi Processor: One processor must be available for each column of the processed images. Thus for 512×512 images a linear array of $P = 512$ processors is required. For wider images several GIPSi chips can be combined to form one wide line. Each single processor contains some local registers, an ALU, local memory and one stage of a shift register (see section 3.2). The combined shift register stages of all processors form a global shift register used for image input and output.

Parameter Memory: The parameter memory is designated for program parameterization and is loaded by a controlling host processor. All processors in the system have read access to this memory simultaneously.

Margin Control: Each processor can communicate to its left and right neighbors for local window processing. However the left and right outermost processors only have one local neighbor on chip. The margin control units allow programmable handling of these margin connections.

Column and row registers: A column and a row register hold the size of the processed images and must be set by a controlling host processor. The values are required for controlling the input/output shift register to detect end of line and end of image conditions.

3.2 Architecture of a single GIPSi processor

Image processing algorithms will be executed by all single-bit processors of the GIPSi system simultaneously. When creating a program the programmer may think in writing it for a single processor but minded that it runs on all processors of the array. Figure 3 shows a single processor. At the top one stage of the input/output shift register SR is shown. Below the local memory and the connection to the global parameter RAM can be identified. Five registers are at the programmers disposal: two memory registers (MR0, MR1), a communication register (CR) and two result registers (RR0, RR1). The ALU is shown at the bottom.

Shift register: The shift register SR is used to move image data into and out of the GIPSi array. Shifting is done in parallel to program execution. Because input and output is done synchronously only one shift register is required. When the input/output shift of a source/result line pair is complete a transfer from SR into the local memory and vice versa is done under program control. Synchronizations between transfer instructions and the SR hardware is done automatically and needs no intervention by the programmer.

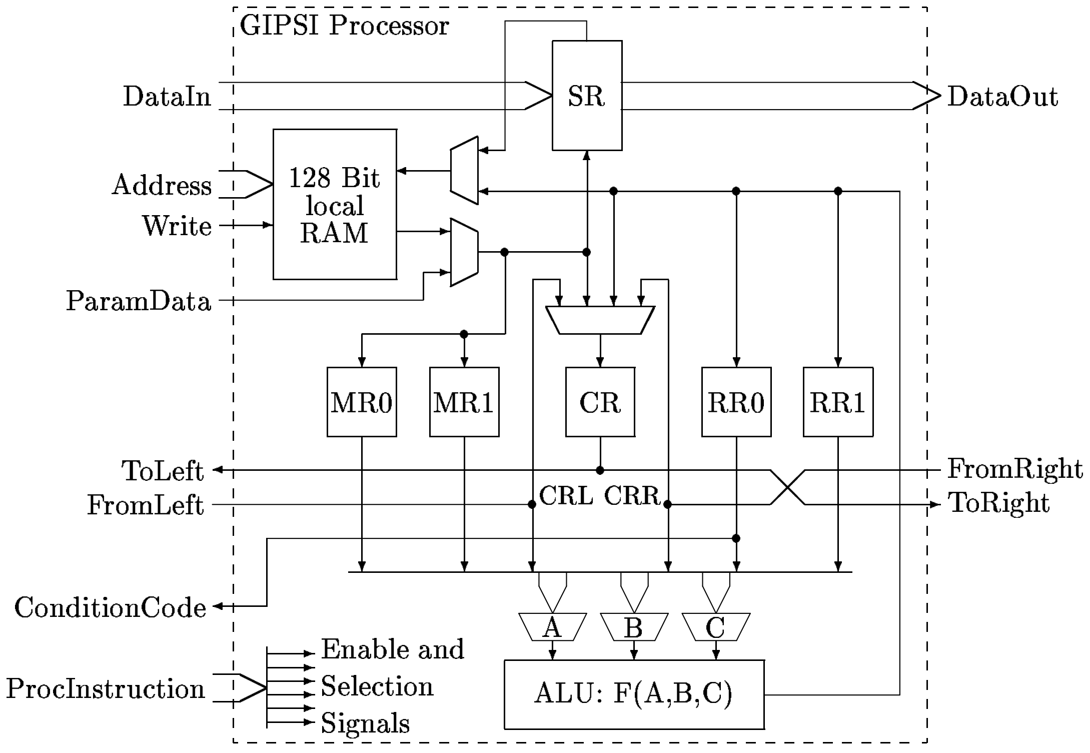

Figure 3: Programmers view to one processor of the GIPSi system

Memory: Each single processor has its local memory of 1 Bit wordsize. The current version offers 128 bits which can be read or written. Besides the local memory all processors have access to the parameter memory introduced above. Local memory and parameter memory are distinguished by different address ranges in a common address space.

Registers: Three types of registers are present. Memory registers `MR0` and `MR1` are loaded from memory and serve as `ALU` source operands. Result registers `RR0` and `RR1` are loaded with intermediate `ALU` results and also serve as `ALU` operands. A communication register `CR` can be loaded from memory or from the `ALU` or via `CRL`/`CRR` from the communication register of the left/right neighboring processor. `CR` further serves as source for the left or right neighboring processors `ALU`.

ALU: The `ALU` maps three source operand bits to one result bit. Any arbitrary function is allowed for that mapping as explained in [Lan94b]. Local registers (`MR0`, `MR1`, `RR0`, `RR1`) or communication registers of direct neighbors via `CRL`, `CRR` can be selected as source. The result can be written into registers `RR0`, `RR1`, `CR` and into the local RAM memory.

3.3 Supporting software tools

Software development is supported by symbolic assembler and a simulator/debugger. The assembler is available for PC and SUN-Workstation platforms, the simulator is available for PCs.

The assembler matches the GIPSi structure. It offers a block oriented language with local scopes for variables and constants. It allows to specify parallel actions which are executable by the processors within one cycle. Conditional instruction specification is supported which eases the creation of parameterizable programs and macros. Macro handling is done by a built-in preprocessor.

The simulator gives a close look into program execution on the GIPSi system. It works on TIFF image data and on instruction sequences generated by the assembler. Single step execution and examination of each processor makes the software development transparent.

4 Algorithm implementations

The following table shows several filters and related execution cycles to compute one result image line on the GIPSi system. The cycle count for full images can be achieved by multiplying the values by the number of image lines.

Table 1: Execution times for image processing operations using GIPSi

Algorithm:	Cycles:
Linear filter with constant coefficients (3x3)	< 1170
Linear filter with constant coefficients (5x5)	< 3135
Min/Max-operators (Erosion, Dilatation) (3x3)	180
Sobel edge detection	252
Kirsch edge detection	1173
Rank order filter (3x3)	538

Assuming a clock rate of 50 *MHz* up to 3200 cycles can be executed during the $64\mu s$ scan period of one video line. The implementations are based one a macro library. This library includes macros for elementary arithmetic operations, various comparisons, communication between neighboring processors, input and output operations, and more.

4.1 Bit-based rank order filtering for VLSI and SIMD systems

A rank order filter will be presented as an example how to prepare algorithms for the GIPSi system. The algorithm is suitable for SIMD and VLSI implementations. It is based on the algorithm of Gu and Swamy [GS92] which was developed for VLSI implementations.

A *rank order filter* of rank l applied to an ordered set $\mathcal{P} = \{p_0, p_1, \ldots, p_{w-1}\}$ of integer values selects one element as result r. In this section rank l is assumed as follows: a filter with rank l selects the l-biggest element. E.g. a rank $l = 1$ filter selects the biggest element (maximum filter) and a rank $l = (w-1)/2$ filter (w odd) determines the median element of a given set.

The algorithm compares the bits of the elements in descending order and thus belongs to the class of *radix sort* algorithms. In contrast to algorithms based on a *divide and conquer* strategy (e.g. [Knu73]) which require dynamic resources this algorithm is static and thus suited for SIMD computing.

Let $p_k[i]$ be bit i of element p_k, $0 \leq i < (n-1)$ where n denotes the wordsize of the elements then the algorithm in Figure 4 selects the value r from set $\mathcal{P}$ with rank l. While scanning the set elements p_k at decreasing bit positions the search status is kept in two vectors f and pp of size w. The bits $f[k]$ and $pp[k]$ reflect the search status of element p_k.

The vector f stores a *"found"* status. If the ordering of an element p_k with respect to the demanded rank value r has been determined then the corresponding bit $f[k]$ is set to 1, otherwise it remains 0. The vector pp holds precalculated bits of the elements for comparison steps. If $f[k] = 0$ then $pp[k]$ contains that bit of p_k which is required for the next comparison step, otherwise $pp[k]$ keeps the bit of p_k from that position where the ordering of p_k respective r had been determined.

Initialization phase: The highest bits $p_k[n-1]$ of all elements are accumulated. The achieved sum s is compared against rank l. If $s \geq l$ then at least l elements have the highest bit set to 1 and are equal or bigger than the demanded rank value r. Thus the rank value r must have its highest bit $r[n-1]$ set to 1. If $s < l$ then at least $w - l + 1$ elements have the highest bit set to 0 and thus $r[n-1]$ must be 0.

Then the *found* status is determined by comparing the result bit $r[n-1]$ against the highest bit $p_k[n-1]$ of each element. If $r[n-1] \neq p_k[n-1]$ then the ordering of p_k respective r is known and $f[k]$ is set to 1, otherwise $f[k]$ is set to 0.

```
$\mathbf{Rank\_Order\_Filter}(\mathcal{P}, l, n) \longmapsto r$
    Initialization Phase:
        $s := \sum_{0 \le k < w} p_k[n-1]$;
        **if** $(s \ge l)$ **then** $r[n-1] := 1$ **else** $r[n-1] := 0$;
        **for** $k = 0$ **to** $w - 1$ **do begin**
            **if** $(r[n-1] \ne p_k[n-1])$ **then** $f[k] := 1$ **else** $f[k] := 0$;
            **if** $(f[k] = 0)$ **then** $pp[k] := p_k[n-2]$ **else** $pp[k] := p_k[n-1]$;
        **end;**
    Main Loop:
        **for** $i = n - 2$ **downto** $1$ **do begin**
            $s := \sum_{0 \le k < w} pp[k]$;
            **if** $(s \ge l)$ **then** $r[i] := 1$ **else** $r[i] := 0$;
            **for** $k = 0$ **to** $w - 1$ **do begin**
                **if** $(r[i] \ne p_k[i])$ **then** $f[k] := 1$ **else** $f[k] := f[k]$;
                **if** $(f[k] = 0)$ **then** $pp[k] := p_k[i-1]$ **else** $pp[k] := pp[k]$;
            **end;**
        **end;**
    Termination Phase:
        $s := \sum_{0 \le k < w} pp[k]$;
        **if** $(s \ge l)$ **then** $r[0] := 1$ **else** $r[0] := 0$;
$\diamond$
```

Figure 4: Radix based rank order filter algorithm using static data structures.

For all elements p_k with $f[k] = 1$ the remaining bits $p_k[i]$, $n-2 \le i \le 0$ are not relevant for further comparisons. Instead bit $p_k[n-1]$ is used for further comparisons and will guarantee that each element p_k once found to be bigger or smaller than r remains bigger or smaller. Thus if $f[k] = 1$ then the precalculated element bit $pp[k]$ is set to $p_k[n-1]$, else it is set to the next bit $p_k[n-2]$.

Main loop: The main loop is executed for all inner bit positions i, $n-2 < i \le 1$. The precalculated bits $pp[k]$ are accumulated. The achieved sum s is compared against rank l to determine the result value bit $r[i]$ as described for the initialization phase: **if** $(s \ge l)$ **then** $r[i] := 1$ **else** $r[i] := 0$. Then follows the update of f and pp. For each element p_k if $r[i] \ne pp[k]$ then the status $f[k]$ is set to 1 because the position of p_k respective r has just been found, otherwise $f[k]$ is kept. If $f[k]$ remains 0 during the update then $pp[k]$ is set to the next element bit $p_k[i-1]$ to prepare further comparisons, otherwise $pp[k]$ is kept because the ordering of p_k respective r is known.

Termination phase: The termination phase is equal to the first part of the main loop with $i = 0$. After evaluating $r[0]$ the requested rank value r is known and the algorithm can terminate without any further updates.

Special care has to be taken when implementing the accumulation step (e.g. $s := \sum pp[k]$). Adding single bits one after another to a sum $s = (s_{m-1}, \ldots, s_1, s_0)$ of fixed size can dramatically slow down the algorithm. A tree decomposition of the summing with reduced word length for intermediate partial sums will remarkable increase efficiency.

Figure 5 shows an example of computing the median of the set $\mathcal{C} = \{15, 6, 10, 5, 6\}$ ($l = 2$, $w = 5$). It follows directly the algorithm of Figure 4.

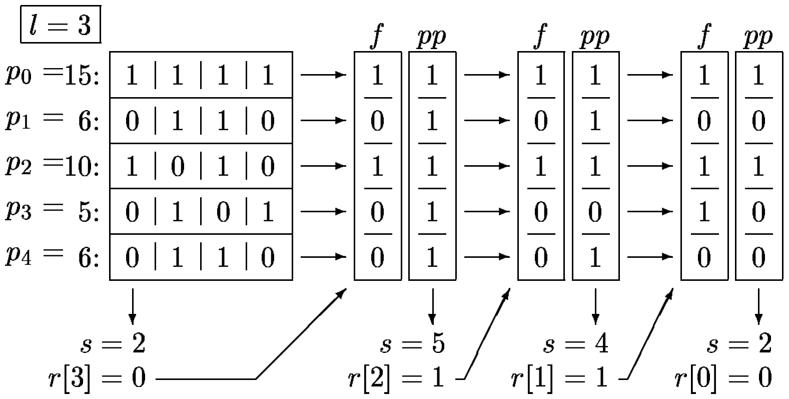

Figure 5: Example: rank order filtering, $\mathcal{P} = \{15, 6, 10, 5, 6\}$, $l = 3$ (median).

5 VLSI implementation

A functional hardware description of the GIPSi system has been developed using the VDHL language. The functionality has been verified using a testbench also written in VHDL. This testbench supplies real image data (TIFF images) and instruction sequences generated by the GIPSi assembler to the functional model of GIPSi.

The ASIC implementation is based on a modern "concurrent design" approach. Starting with the the functional description sucessive partitioning, floorplanning and synthesizing steps lead to a first structural description of the system. This first description yields first physical parameters with high accuracy. Critical parts in the design can be detected. Based on these first parameters the structural description is iteratively refined. Regular structures in the design are elaborated by means of data path and memory compilers. For underlying cells portable libraries are used throughout the implementation which ease the change towards improved technologies.

A first GIPSi cell layout based on $0.6\mu m$ structures has been achieved. The underlying refined structural chip model has been has been tested using the data from the functional verification (TIFF images, GIPSi instruction sequences). Preliminary parameters of the $0.6\mu m$ layout are:

- 25.6×10^9 bit operations per second.
- 64 kBit on-chip SRAM.
- silicon area: ca. $120\, mm^2$.
- system frequency: $50\, MHz$.
- 144-Pin ceramic PGA package.
- 512 processors on chip
- logic complexity: 132 000 gates.
- 5 Volt processor.
- $0.6\, \mu m$ CMOS technology.

6 Conclusions

The GIPSi system has been presented which is a parallel VLSI system designed for image processing of line-scan image data. Basic low level algorithms have been implemented which show very short execution times. A complex rank order filter suitable for SIMD processing has been presented in detail.

These first implementations show that the GIPSi system architecture is flexible and can handle various different linear and non-linear image preprocessing algorithms in real time.

Further investigations will analyze other algorithms e.g. from the image compression and digital TV domain and extend the current architecture and thus the number of applicable algorithms.

References

[Ste83] Stanley R. Sternberg. Biomedical Image Processing. *Computer*, Seite 22–34, January 1983.

[Ree84] Anthony P. Reeves. SURVEY Parallel Computer Architectures for Image Processing. *Computer Vision, Graphics and Image Processing*, 25:68–88, 1984.

[Kun88] S.Y. Kung. VLSI Array Processors Prentice Hall, Englewood Cliffs, NY, 1988.

[FH85] A.L. Fisher and P.T. Highnam. Real-Time Image Processing on Scan Line Array Processors. IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database Management, Miamy, FL, 1985.

[Wil88] S.S. Wilson. One Dimensional SIMD Architectures — The AIS-5000. In S. Levialdi, editor, *Multicomputer Vision*, Academic Press, 1988.

[MKW$^+$90] H. Miyaguchi, H. Krasawa, S. Watanabe, J. Childers, P. Reinecke and M. Becker. Digital TV with Serial Video Processor. *IEEE Transactions on Consumer Electronics*, Vol. 36, No. 3, August 1990.

[CS91] K. Chen and C. Svenson. A 512-Processor Array Chip for Video/Image Processing. In H. Burkhardt, Y. Neuvo and J.C. Simon, editors, *From Pixels to Features II, Parallelism in Image Processing.* ESPRIT BRA 3035 Workshop, Bonas, September 1990.

[YKF94] N. Yamashita, T. Kimura, Y. Fujita, Y. Aimoto, T. Manabe, S. Okazaki, K. Nakamura, and M. Yamashina. A 3.84GIPS Integrated Memory Array Processor LSI with 64 Processing Elements and 2Mb SRAM. ISSCC94, Session 15, Paper FA 15.2, 1994

[Lan94a] B. Lang. Self Arbitrating Elements for Modelling Systolic Dataflow in Field Programmable Gate Arrays. In *Proceedings GI/ITG Workshop Anwenderprogrammierbare Schaltungen*, Karlsruhe, 1994.
Available via Internet: `ftp://www.ti1.tu-harburg.de/pub/papers/la:ka.ps`

[Lan94b] B. Lang. The GIPSI System. Internal Report No. 9/94, Technische Universität Hamburg-Harburg, Technische Informatik I, 1994.
Available via Internet: `ftp://www.ti1.tu-harburg.de/pub/papers/la:gipsi.ps`

[GS92] Q. Gu and M.N.S. Swamy. A Binary Logic Synthesis Approach to the Bit-Level Implementation of Generalized Rank-Order Filters. 1992 IEEE Intl. Symposium on Circuits and Systems. Vol. 1, San Diego, CA, May 10-13, 1992.

[Knu73] D.E. Knuth. The Art of Computer Programming, Vol. 3: Sorting and Searching. Addison Wesley, 1973.