Real time object tracking with a low-cost smart camera

Yannick Morvan
Richard P. Kleihorst
Anteneh Abbo
Philips Research Laboratories,
The Netherlands.

Harry Broers
Peter Raedts

Philips Centre for Industrial Technology,
The Netherlands.

Abstract

Reliable object tracking at video speed is a task that requires high performance. We have analyzed the various parts of typical algorithms and found that some parts of program flows are best performed on a parallel platform for performance reasons and other parts are better performed on a sequential processor for reliability and versatility reasons. Combining the two processing profiles in one platform gives the best of both worlds and enables real time, reliable performance. To prove this concept, we have designed and built a programmable architecture consisting of a massive parallel pixel processor and a sequential DSP. On this platform we mapped the challenging task of high speed object tracking. It was found that offloading the pixel intensive tasks to the parallel processor gives ample processing power to make reliable vision decisions and environment control on the sequential processor. Good results are demonstrated for robot based soccer playing in the RoboCup environment.

Keywords

Image processing, smart camera, parallel procesing, blob tracking.

1. Introduction

Computer vision is getting used in more and more applications in small fabrication plants, household applications and gaming. The reason for this to happen is the increasing integration feasibility of hardware and software in the latest silicon technologies where the system is integrated on preferably one, but not more that a few chips.

If one looks at the typical performance level necessary for even simple systems the demands are still impressive. Especially if the tasks are to be performed in real time, for instance at 30 frames per second with VGA resolution, an operative power reaching GOPs is easily claimed. Contrary to popular belief, this is not simply due to the complexity of the algorithms. It is more due to simple steps in the algorithmic flow such as edge detection, morphological operations and pixel segmentation. Although these steps are simple in complexity, an immense number of pixels has to be processed. In our 30 fps VGA case, which is actually low frequency and small image size, almost 10M pixels per second flow through the system. So, a small amount of 100 instructions on each pixel already relates to 1 GOPS.

Typically, these high numbers are only found in the low-level imaging tasks of the chain. At the next (high and intermediate) level, the algorithms get more complex but less data items are processed per second. Here, we work on objects, parameters or edges.

Having a closer look at the two levels inside algorithms, pixel and object processing, we discover that except from differences in data-granularity, there is also a difference in thread similarity. While for two different objects found in a scene two different high-level tasks can be performed, for two pixels in the scene exactly similar tasks are performed. This observation closely matches to the two-processor concepts of VLIW (very large instruction word), and SIMD (single instruction, multiple data). On VLIW processors, larger complex programs can be easily mapped, while on SIMD processors simple tasks can be performed in parallel up to high effective operational speeds [1, 2].

So, mapping the low-level task on an SIMD processor followed by a VLIW processor to do the high level tasks seems to be a good option. Here, the high performance in GOPS needed is now provided by the immense power of a pixel parallel processor. This offloads the demands from the VLIW processor, which can now completely be dedicated to the more complex items in the high level tasks. Because of this division a higher throughput is obtained and real-time applications are possible. Key issue here is the data communication between the two processors, if the VLIW processor is looking for data on a polling or interrupt basis, the amount of data should be as low as possible. In

other words, most of the segmentation has to be performed in the SIMD processor.

In this paper we show the aforementioned concept of processor and process partitioning for the task of object tracking in a gaming environment. In the well known RoboCup game, autonomous robots are made for playing soccer [3]. The different items of interest in the field such as ball, goals, field and competitors are identifiable by color. As an experiment to investigate our concept we mapped the pixel color segmentation and morphological operations on the SIMD processor and the high level decisions, such as robustness and robot control on the VLIW processor. The two processors are the heart of what we call our hybrid camera. In this system control results are now made on input video speed.

In Section 2, we discuss our system consisting of the SIMD and VLIW processors. In Section 3, we discuss the algorithm used to detect objects in the field. In Section 4, we show some results of the system and Section 5 draws the conclusions.

2. Hybrid Camera Architecture

The proposed camera architecture is shown in figure 1. The front-end (pixel processing) is the computationally intensive part and is conducted on the Xetal parallel processor [4]. The logical place of this processor is immediately after the image sensor so that basic pixel processing tasks can be performed directly at the chosen pixel rate without the need to store in the frame memory. Output(s) of the pixel processor is transferred to the TriMedia [5, 6] block processor. TriMedia is a general-purpose media processor with a capability executing up to five instruction in parallel.

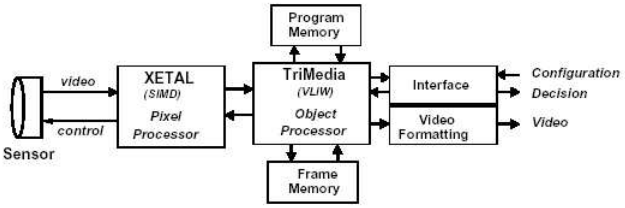

Figure 1: Hybrid Camera Architecture

The synergy between parallel and sequential processing is exploited fully when the application is properly partitioned into tasks that efficiently execute on the respective processors. In our approach, the algorithms are statically partitioned and each processor executes its part of the job with video signals used both as data and synchronization. The program memory in Figure 1 holds the code segments for the TriMedia and Xetal processors. After system reset, the Trimedia transfers the Xetal code to the on-chip program memory and sets some configuration parameters. It is also possible to update the parameters on-line to achieve some sort of closed-loop video processing.

In the proposed architecture, the TriMedia executes a real-time operating system and is responsible for delegating tasks to the parallel processor and other peripherals. This is in addition to the video post-processing needed to interpret the outputs of the parallel processor and generate appropriate decision signals to control/serve the host machine/robot.

2.1 The Xetal Parallel Procesor

The top-level architecture of Xetal is shown in 2. There are two programmable digital processors: a Global Control Processor (GCP) for the line-based algorithms and a Linear Processor Array (LPA) consisting of 320 identical processing elements for pixel-based algorithms. Both processors obtain their instructions from the same program memory which has been filled by the TriMedia during system start-up. The LPA can use 16 line memories for temporary data storage. In addition to the GCP and LPA, Xetal has a parametrizable serial processor which performs tasks like video-formatting, region-of-interest selection and statistical computations. The I2C protocol is used for interface with other ICs.

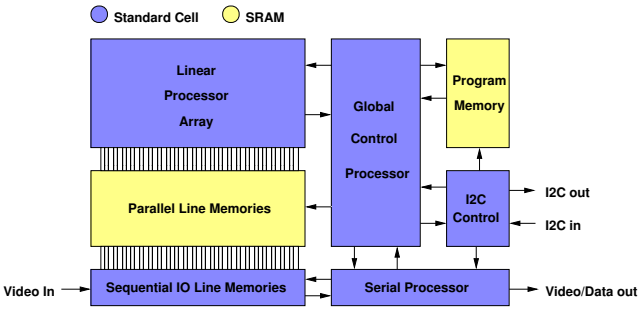

Figure 2: Top-level architecture of Xetal

The detailed architecture of the LPA is shown in. The signal paths are all 10 bits signed-magnitude. Each processor contains a 10-bit accumulator that stores the last result, which can be used as an operand for the next instruction. Both an adder and a multiplier are implemented in the ALU and with these functions comparison, addition, subtraction, data weighing and multiply-accumulate are performed within one clock cycle. By limiting the data weighing to coefficients between -1 and 1, no over or underflow in the 10 bit data-range will occur.

Data-by-data multiplication is not possible in the LPA which, of course, limits the implementation of some specific algorithms. However, the processors

incorporate a 1-bit flag that is set according to the latest result. Based on this flag conditional pass-instructions are possible, allowing a limited form of data-dependency in the algorithms. All 320 flags are connected to a global line which can be seen and reacted upon by the Global Control Processor. In this way, iteration processes with a certain stop condition can be run on the parallel processor or, in other words, Xetal can react on image contents.

By using a multi-port principle for the line-memories, in one instruction-cycle operands can be read from memory and results can be written back at desired locations. From the line-memories, 16 of the 19 words are used as regular RAM area for signal processing purposes. The remaining three are write-only memories for storing the final results of the calculations, which are then read out sequentially by the serial processor. For most of the algorithms, the results will be YUV or RGB signals; however, image description data is also possible. The interface to the video source is achieved via sequential line-memory that is serially written and read out in parallel by the LPA.

Tasks that the parallel processor is used for in our application area include FPN reduction, defective pixel concealment, noise reduction, color reconstruction and color domain transformation, including pre-filtering for sub sampling purposes. As this processor is programmable, custom programs can be made for the above tasks. Also numerous other tasks are possible, including template matching, segmentation, color keying, event recognition and even simple forms of image compression.

The global control processor is responsible for the synchronization of the entire chip. Its main task is to update the program counter, to fetch and decode instructions and pass them to the LPA. Besides that, it can do global calculations for exposure-time control, white-balancing and such by using the statistical information which is updated (by the serial processor) in internal registers. A number of logical and arithmetic instructions are available in the global processor including multiplication. For program control, the global processor handles (conditional) jumps in the program and therefore contains a number of (program) counters. In addition, the global processor can react on events generated by the parallel processor, and jump to different subroutines. Certain registers of the global processor are accessible through I2C and therefore give a way of communication to and from the end user. For instance, a region of interest can be specified to the controller, which then addresses the selected rows and columns only for processing.

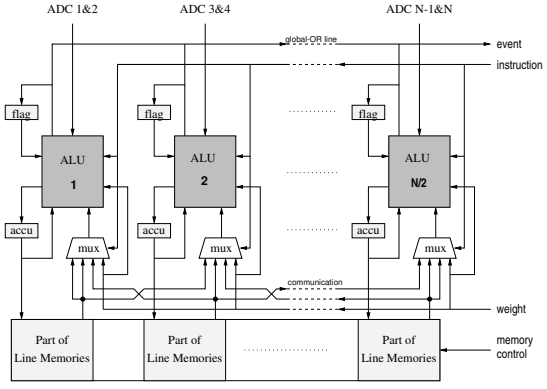

Figure 3: The linear processor array with communication lines.

The adopted global processor architecture and single memory coding style avoid the instruction (24-bits wide) bandwidth problem encountered with other low-cost microcontrollers such as MIPS PR1900 CPU [7]. Although these general processors come with tools for controller code development in high-level descriptions, the fact that the Xetal DSP code dominates the total program and still has to be manually written lessens the advantage. For this reason we have incorporated a custom designed global controller, which could meet the Xetal instruction bandwidth requirement.

2.2 Programming Framework

One of the challenges of a multi-processor platform, like the hybrid camera architecture proposed, here is the ease and efficiency with which algorithms can be developed and mapped. Attributing to its general-purpose architecture, the TriMedia comes with a complete set of software development tools that allow efficient code development starting from the C language. The Xetal processor, on the other hand requires careful programming discipline to exploit the processing power to the full. Vision algorithms need to be properly written with the parallelism of the final architecture in mind. To assist program development, Xetal comes with a software framework having an assembler, a simulator and a debugger. The Xetal program is written in a C-like assembly code. The development of a unified programming platform to facilitate or possibly automate the partitioning of vision tasks in the proposed hybrid architecture is under investigation.

3. Implementation of color based segmentation

In this section we will describe the mapping of the algorithm on our hybrid system. The algorithm consists of the various typical parts used in color-based object tracking: identification, noise removal, coordinating and using the result. As the programs on the TriMedia VLIW processor are straightforward and the programs on Xetal are more exotic, we will focus more on the latter.

3.1 Colour segmentation YUV vs. HSI

Because the object, in this example, the ball can be identi- fied based on color, we implemented a program to detect objects by color matching. To be independent of illumination changes in the RGB domain, an intensity independent color space has been used. Two such color space transformations have been implemented in Xetal: the RGB to YUV and RGB to HSI (Hue, Saturation, Intensity) color space. The computation of the first simple and linear one is discussed in [8]. Because of the non-linear nature of the second one, the implementation of the HSI transformation is more difficult. Nevertheless, the use of HSI is preferred in color segmentation because it differentiates the color intensity component from the chromatic component. Moreover, it also differentiates pure color (hue) from saturation. So it is possible to segment the image using only one parameter value which makes the segmentation more robust. Consequently, tuning to different colours will be easier. The HSI transformation implemented is shown below:

$$H = \frac{1}{2p}\{a - \arctan[R - I\frac{\sqrt{3}}{G-B}]\}$$
$$S = \sqrt{(R^2+G^2+B^2)-(RG)-(RB)-(BG)}$$
$$I = \frac{R+G+B}{3}$$

Where only H is calculated and matched.

3.2 Erosion

To make the color-based detection more robust, we implemented morphological erosion [9]. This operator removes noise and other small blobs. Because the color segmentation provides a binary image, erosion fits perfectly the flow algorithm. Moreover, the Xetal architecture is suitable for this kind of operators which can be implemented very easily.

The erosion algorithm sets the output pixel to one if all its neighbors are equal to 1. To achieve this, a kernel of 3 by 3 is applied.

The computation of this operator requires the allocation of three lines memories in Xetal. The output of the color segmentation provides a line composed by zeros for the background pixels and one's for the foreground (the red ball). The first result is stored in the first temporary line. Then, thanks to the local communication between each processors elements, the kernel is applied by adding the current pixel with its local neighbors (available in the two other temporary lines). If the result of the addition for the current pixel is less than 9 (3 by 3 kernel of one's), the pixel is set to zero, else it is set to one. When a new line is acquired, the two previous lines are shifted into the next temporary lines and the new one is loaded in the first line.

PID line	1	2	3	4	5	6	...	320
line_max	0	0	52	...	84	0	...	0
line_min	...	511	52	...	84	511	...	511

Table 1: The line-memories line_min and line_max

3.3 Feature extraction

The extraction of the position of the object in the image is performed after color segmentation and erosion (a binary image).

To achieve the localization of the object, the detection of the first row, last row, first column, last column of the object is good enough. Thanks to this information, we can draw a marking square around the object using the TriMedia processor for demonstration purposes as shown in Figure 6. Since the rows are read sequentially, the detection of the first and the last row is easy. The presence of the object in the current line is detected by the result in the accumulators. The first row is detected when the top line of the object is found. The last row is found when the last line is detected (see Figure 4). If all the accumulators are set to 0, the object is not detected. The instruction JUMPWO allows a conditional jump in this case. An internal row counter provides the identity of the current row.

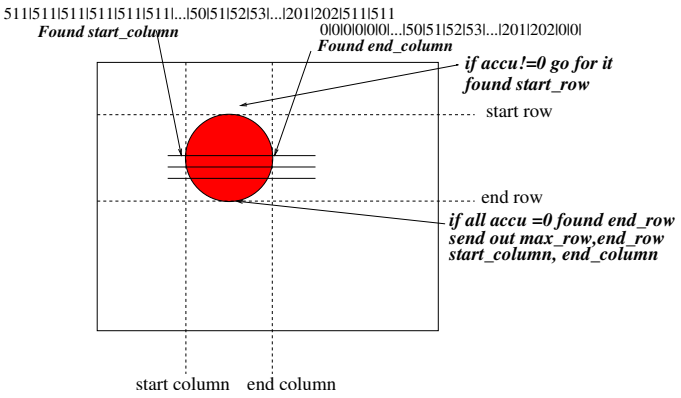

Figure 4: Red ball in the bounding box

The computation of the first and last column is less straightforward. In particular the lack of information (especially the address of the PE) about each processor element due to SIMD structure requires the computation of the processor identity (PID) during the initialization. This procedure fills a line memory by values 1 to 320 (see Table 1). It allows to identify each processor element. This feature allows us to write data on a particular pixel, and locate data in the line memory. The processor ID is computed once at the initialisation phase, then the result remains in one of the 32 line-memories.

To extract the first and last columns of the ball in the line memory, we use the Serial Processor (See figure 2). It provides statistical information on the video stream. In this way, we are able to read maximum, minimum and average values of each channel.

The line provided by the erosion is masked with the PID line to provide two lines: line_min and line_max (see Table 1). The maximum of line_max is the last column of the object and line_min, the first column. Those two lines are sent out on two different channels and the serial processor computes the desired maximum and minimum. These two extreme values correspond to the first column and last column of the ball in the current line.

The maximum is then loaded in the end column, and the minimum in the start column register. The operation is repeated for each line where red pixels are detected, so that the end column and start column are updated every line. To determine the extreme values of the boundary box, the routine keeps the smallest first column and the biggest last column provided by the serial processor.

When a blank line is detected, the end of the object is reached, the program sends out the data (the coordinates of the boundary box) via the communication channel. To send out the data in an efficient way, we chose to insert the coordinates in the video stream. The coordinate data is located in the left most pixels of the line memory. The rest of the line memory is filled with the regular video stream.

The protocol adopted to transfer the coordinates to the TriMedia is illustrated Figure 6. It allows the TriMedia to receive the video stream and to get the coordinates of the object with just one channel. To minimize data transfer, we send just the 4 bytes (coordinates of the boundary box) to the Trimedia. The objective is to achieve a large compression ratio to enable the TriMedia to spend more time to high level imaging tasks instead of input port polling or interrupt requests.

For one object in the image, the ratio of compression achieved is 4/(640*480).

4. Results

The computation of color segmentation requires 20 percent of the Xetal capacity.

The equation of the erosion operator can be directly ported to Xetal. Moreover, the parallel architecture of Xetal allows a very fast achievement of this calculation: it requires only nine clock cycles each line and nine more to set or reset the corresponding pixels. This operation fills 1.7 percent of the instructions doable in real time by the DSP and only 10 percent of the memory available. This routine can be called several times but requires three lines memories more for each additional stage.

The result of the color segmentation and the erosion operator is shown in the lower part of Figure 5:

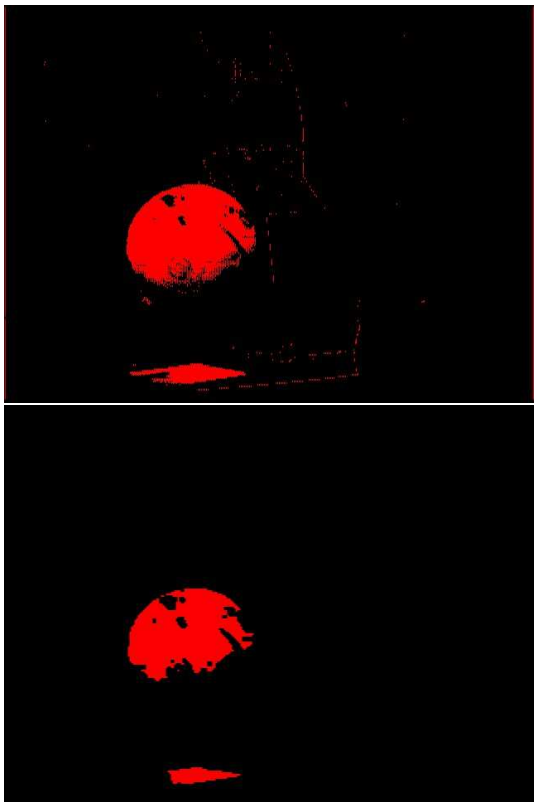

Figure 5: Result of the erosion operator

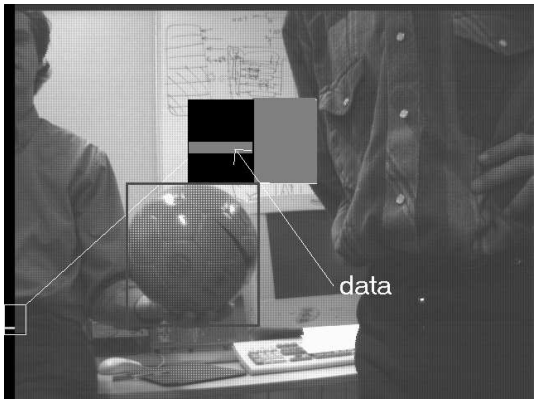

Figure 6: Final result on the display (the coordinates are on the left below the last line of the ball)

The method adopted for the tracking enables to detect several red objects in the image. At a resolution of 640*480 and a 30 frames per second, the system uses 56 percent of Xetal. The output image is shown Figure 6, where we used the TriMedia to indicate the bounding box found and to control the robot.

5. Conclusions

In this article we have mapped a computationally intensive algorithm on suitable processors: an SIMD (parallel) processor for the low-level imaging tasks and a VLIW (sequential) processor for the intermediate and high-level tasks. We have shown that this hybrid system is able to perform this task in real time, at camera speed. Moreover, half of the system is free to do other tasks in the system. Because this solution basically consists of only two chips, computer vision tasks are becoming closer to be used for non-professional consumer tasks. Namely, in the consumer market, not only performance and robustness, but mainly price determines the success of a product.

References

[1] H. Corporaal, *Microprocessor Architectures : From VLIW to TTA*. John Wiley Son Ltd, 1997.

[2] P. Jonker, *Morphological Image Processing: Architecture and VLSI Design*. Kluwer Technische Boeken, 1992.

[3] "The robocup web site," http://www.robocup.org, 2002.

[4] R. Kleihorst *et al.*, "Xetal a low-power high-performance smart camera processor," in *Proc. ISCAS 2001*, (Sydney, Australia), 2001.

[5] "Portfolio," in *Trimedia Technologie*, http://www.trimediatechnologies.com/, 2002.

[6] "Datasheet," in *Trimedia Technologies*, http://www.semiconductors.philips.com/acrobat/datasheets/75006379.pdf/, 2002.

[7] M. T. Inc., *"The MIPS CPU family."*. http://www.mips.com/, 2002.

[8] R. Kleihorst, M. Lee, A. Abbo, and E. Cohen-Solal, "Real time skin-region detection with a single-chip digital camera," in *ICIP 2001*, (Thessaloniki, Greece), Oct. 2001.

[9] J. Piper, L. Ji, and J. Tang, "Erosion and dilation of binary images by arbitrary structuring elements using interval coding," *Pattern Recognition Letters*, no. 9, pp. 201–209, 1989.