# DATA-DRIVEN ARCHITECTURES FOR IMAGE PROCESSING.

M. P. Eikendal, F van der Heijden.

Twente University
Department of electrical engineering (BSC),
Postbox 217, 7500 AE Enschede, The Netherlands.

## RÉSUME

Une architecture pour le traitement d'images avec le NEC µPD7281 dataflowprocessor a été realisée. La performance est determinée avec des algoritmes primitives (convolution etc.). Le traitement d'images en temps réel est possible tant quelques restrictions sont toujours imposés.

## SUMMARY

The NEC µPD7281 dataflowprocessor is well suited for application in a low-cost multi processor image processing system. When used in a ring structure however, the transport capacity of the ring limits the processing power of the system. This limitation is solved by using a number of processor rings in parallel. The performance of the resulting hardware is evaluated using some basic image processing algorithms. To some extend real time image processing is possible.

## 1. Introduction.

A crucial point in the design of image processing architectures is the trade-off between throughput and flexibility. A possibility to cover both aspects is the use of a number of fast general purpose signal processors in parallel. Such a parallel architecture can be realized in a compact way and at low cost using the NEC µPD7281 dataflowprocessor. This processor is based on a data-driven architecture. In conventional von Neumann processors operations are executed in a predefined sequence, following an operation list (control flow). In contrast to this an operation in a data-driven system is executed upon arrival of the input data for this operation, so the operation sequence is defined by the data dependencies between the operations. Hence dataflowprograms are usually described by a flowgraph: a directed graph where the nodes denote operations and the arcs denote data dependencies between these operations. Data values are carried on tokens that flow along the arcs. When a token is available on each input arc a node is executed and a token containing the result data is produced on the output arc (figure 1).
In the µPD7281 dataflowprocessor, the program is stored in two memories (RAM): the link table, containing the data dependencies i.e. the arcs of the flowgraph and the function table, containing the operations (figure 2). A third memory, the data memory, is used for storage of tokens until the second token for a two input node arrives and for storage of program variables. The token matching and the address generation for the variables is controlled by the address generator & flow controller. The three memories, the processing unit and a queue are placed in a circular pipeline. This pipeline can be passed in 7 cycles of 200 nsec each when the queue is empty (10 MHz. clock). All instructions with a single output token are executed in one pipeline cycle. This includes powerful instructions as a multiply instruction and a multiple bit shift. The queue is added because instructions with multiple output tokens take more than one pipeline cycle.

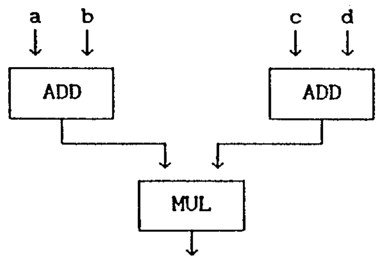
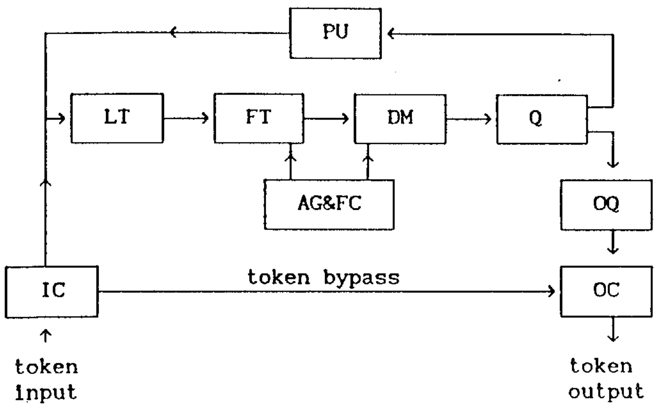


*Figure 1. An example of a flowgraph: computation of (a + b) * (c + d).*

For communication between the µPD7281 and its environment, a token input bus and a token output bus are available. Through these 16 bit busses, the 32 bit tokens are transported in two steps. As the



| | | | |
|---|---|---|---|
| PU = processing unit | AG&FC | = | address generator & flow controller |
| LT = link table | IC | = | input controller |
| FT = function table | OQ | = | output queue |
| DM = data memory | OC | = | output controller |
| Q = queue | | | |

*Figure 2. The internal structure of the μPD7281.*

format of the processor input tokens, the processor output tokens and the tokens leaving the processing unit is the same, processors can easily be cascaded. To enhance the possibilities for cascading processors, a token bypass between the input controller and the output controller is added. During reset a 4 bit module number is assigned to each processor. The input controller sends tokens with a matching module number to the link table. All other tokens are passed directly to the output controller.

## 2. The basic architecture of a flowprocessor system.

As mentioned before, processors can easily be cascaded. A processor-ring offers the possibility of communication between all processors. Two other units needed in this ring are:
- A host interface for downloading the dataflowprogram by sending special table-write tokens and for data input and output.
- A memory interface. The data memory of the μPD7281 is only 512 words. This is insufficient for most image processing so external memory must be added. This memory can be accessed by sending memory READ or WRITE tokens to the memory interface.

NEC has developed a chip for these functions: the MAGIC (memory access and general bus interface chip). Furthermore the MAGIC has provisions for adding a DMA-controller through which the flowprocessors can access the host memory (figure 3).

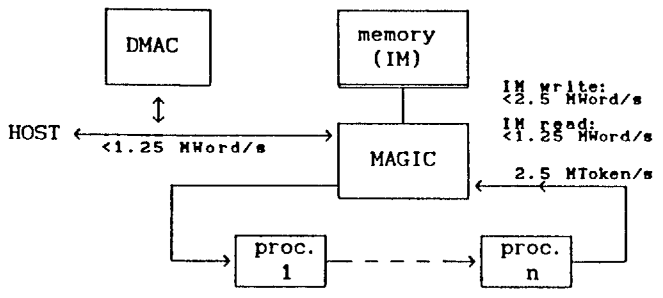


*Figure 3. The basic architecture.*

The performance of this architecture is limited by three factors:
- The processing power of a processor. It is possible to overcome this limitation by increasing the number of processors (maximal 8).
- The transport capacity of the inter-processor network. This network is used by all processors to access the external memory.
- The transport capacity of the host interface. Data transport is under flowprocessor software control. This limits the transport capacity (1.25 MWord for host → external memory, 0.8 MWord for external memory → host).

In the next paragraph the effect of these limitations on the performance of the basic architecture is evaluated using some basic image processing algorithms.

## 3. Performance of the basic architecture.

The performance of the above architecture is evaluated using the following image processing algorithms:
- histogram. Computation of a 8 bit histogram of pixel values. The histogram is stored in the flowprocessors data memory and the input image in the external memory.
- monadic operation. The value of a pixel in the input image is transformed to a value of an output pixel using a look up table in the processors data memory.
- dyadic operation. The value of the output pixels is the result of an operation on the corresponding pixels in two input images.
- convolution I. A fast algorithm optimized for a 3x3 convolution kernel.
- convolution II. This algorithm is not as fast as the first convolution but its kernel size can be varied to a maximum of 13x13.

Two approaches to run these algorithms on more than one processor are available:
- Algorithm partitioning: each processor performs a part of the algorithm on all input data.
- Data area partitioning: each processor performs the entire algorithm on a part of the input data.

The first approach asks special attention for balancing the tasks of the processors to acquire optimal use of all processors. Besides this, some synchronization between the processors is needed to prevent overflow in the processor with the heaviest task. Hence the second approach is chosen for the above algorithms. This approach is well suited for image processing as many algorithms perform the same operation on each pixel. No synchronization between the processors is needed and the processing power of the system is to some extend proportional to the number of processors in the ring. In table 1 the execution times of the first four algorithms measured on a single processor are given (256 x 256 images). These times can be divided by the number of processors in the ring to obtain the execution time for the overall system. However the number of useful processors in the ring is limited by the capacity of the inter-processor token ring. The minimum execution time due to this limitation is given in the third column of table 1. It is estimated from the number of tokens sent to the MAGIC for image memory access. A memory read token requires in general one token and a memory write two tokens. For example the convolution algorithm reads a column of 3 pixels from the input image for each output pixel so 5 tokens are sent to the MAGIC per pixel. This occupies the token ring for 2 μsec per pixel or 130 msec for the entire image. Comparing this value to the execution time for one processor leads to a maximum number of 7 useful processors for the convolution algorithm. The values for the other algorithms are estimated in a similar way (table 1).

*Table 1. Execution times on a single processor and on an optimal number of processors for 256 x 256 images.*

| operation | execution time 1 proc. (msec) | minimal ex. time (msec) | number of useful processors |
|---|---|---|---|
| histogram | 150 | 26 | 6 |
| monadic op. | 200 | 78 | 3 |
| dyadic op. | 290 | 104 | 3 |
| convol. I | 800 | 130 | 7 |

The above execution times are measured during program execution on the actual hardware. A method to obtain the execution time of an algorithm in an earlier stage of program development is demonstrated for the convolution II algorithm. The execution times of this algorithm are estimated using the following formula:

$$t_{ex} = a * \frac{N^2 * t_c}{e_{LT} * n}$$

with:

$N^2$ = image size i.e. the number of pixels.
n = number of processors.
$t_c$ = pipeline cycle length (200 nsec at 10 MHz.).
$e_{LT}$ = Link Table efficiency i.e. the percentage of time the Link Table is used. Simulations show that the LT efficiency is almost independent of the kernel size ($e_{LT}$ = 80 ± 5%).
a = number of tokens passing the Link Table per pixel. This value can be derived from the flowgraph by counting the number of output tokens per node. For a p x p convolution kernel:

$$a = 3p^2 + 8p + 30$$

Substitution using N = 256 results in:

$$t_{ex} = (3p^2 + 8p + 30) * \frac{16.4}{n} \text{ (msec)}$$

In general a well organized program will have a LT efficiency between 70% and 90% so without simulation an acceptable indication of the execution time can be derived directly from the flowgraph.

For each pixel in the output image, a column of p pixels is read from the input image and one pixel is written. This results in a minimum execution time due to the transport capacity of the token ring:

$$t_{min} = (p + 2) * N^2 * 400 \text{ (nsec)}$$

Comparing this value and $t_{ex}$ results in 10 useful processors for p = 3 and even more for p > 3. The limiting factor for the convolution II algorithm never is the transport capacity of the token ring as the maximum number of flowprocessors in combination with the MAGIC is 8.

Besides the convolution II algorithm the above figures show that for relative simple algorithms the transport capacity of the processor ring is the limiting factor. Hence the number of processors per ring in a general purpose image processing architecture should be limited.

Apart from the capacity of the token ring, another reason exists to limit the number of processor in the ring. In general a memory access requires two tokens to be sent to the MAGIC. When two or more processors are accessing the memory at the same time, care should be taken to prevent the memory access tokens of both processors from getting mixed up. The MAGIC has provisions to make the simultaneous memory access of maximal four processors possible so if more than four processors are used, the memory access of these processors must be synchronized by software.

*Table 2. Execution times for the convolution II algorithm (4 processors, 256 x 256 images.*

| kernel size p x p | LT tokens a | execution time $t_{ex}$ (msec) |
|---|---|---|
| 3 x 3 | 81 | 332 |
| 5 x 5 | 145 | 594 |
| 7 x 7 | 233 | 954 |
| 9 x 9 | 345 | 1410 |
| 11 x 11 | 481 | 1970 |
| 13 x 13 | 641 | 2630 |

As adding more processors in a ring offers no general applicable solution to increase the processing power another alternative is given in the next paragraph.

## 4. Processor rings in parallel.

As demonstrated above the processing power of the ring is limited in some applications by the capacity of the µPD7281 input/output busses. The only way to overcome this limitation is the use of a number of these busses in parallel. This can be realized without leaving the processor ring concept by using a number of processor rings parallel. To prevent the access to the external memory from becoming the limiting factor, each ring should have its own (local) memory. A shared memory can be added for communication between the rings and for distributing the input and the output data. To realize this parallel ring architecture, a VME compatible processor board is developed (figure 4). This board consists of:

- 4 flowprocessors.
- a VME interface with DMA-controller and interrupter facilities.
- "local" memory ( 1M x 18 ).
- an interface to a special bus through which a number of processor boards can be connected to shared memory.

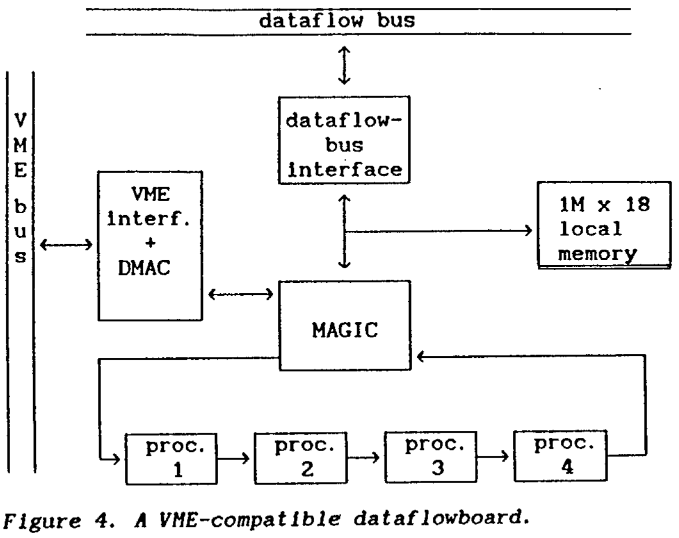


*Figure 4. A VME-compatible dataflowboard.*

The processor board can be used as a stand alone board to increase the processing power of a VME system, but it is also possible to combine several boards together with a shared memory and for instance a dual ported image memory (figure 5). The addition of the dual ported image memory solves the third limitation of the basic architecture: it is no longer necessary to use the relative slow host interface for image data transport.

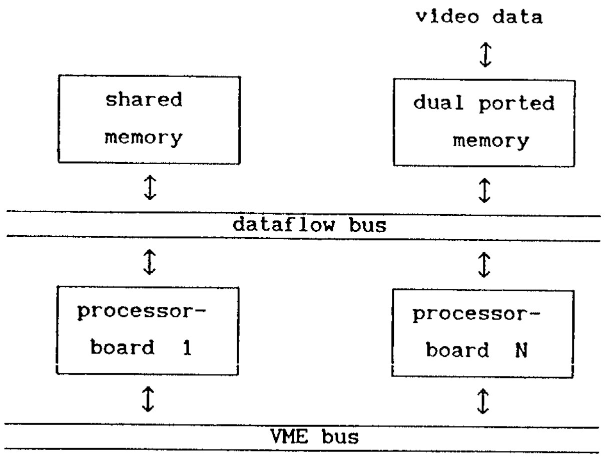


*Figure 5. A realization of a flowprocessor system with parallel processor rings.*

## 5. Real time image processing.

When the data partitioning method is used to divide the algorithm over the processor rings, processing power is proportional to the number of rings. In table 3 the execution times for the four algorithms mentioned before are given for a single processor ring with four processors. This leads to the number of rings (R) needed for real time image processing in the last column of the table (40 msec / image).

*Table 3. Number of rings needed for real time image processing (N = 256).*

| operation | execution time single ring (msec) | R for real time operation |
|---|---|---|
| histogram | 26 | 1 |
| monadic op. | 78 | 2 |
| dyadic op. | 104 | 3 |
| convolution I | 200 | 5 |

The main disadvantage of the above architecture is the fact that additional transport between the different memories can seriously degrade these figures. The transport capacity of the dataflowbus is determined by the relative low speed of the MAGIC memory interface and some overhead for the bus arbitration. Distribution from the image memory to the local memories of the different rings takes as much as 50 msec so together with the collecting of the result data about three frame times are used for data transport. Hence to perform real time image processing a hardware distribution system should be added. Two realizations for this system are:

- For each ring an image memory with facilities to select a part of the total image from the video bus.
- A token based distribution system similar to the inter-processor token ring. Using a special interface with FIFO buffers in each processor ring the required speed for real time image processing comes within reach.

## 6. Conclusions.

A flowprocessor architecture consisting of a number of parallel processor rings with hardware distribution of video data is capable of performing real time image processing algorithms. Using the 10 MHz dataflowprocessor the image size is limited to 256 x 256 but when the new 20 MHz. CMOS version is used real time processing of 512 x 512 images will be possible. This 20 MHz. processor will be available soon.

The above image processing algorithms can of course also be performed by dedicated hardware such as a convolver and a two dimensional look up table. However as general purpose signal processors are used the applications are not limited to low level image processing algorithms. These algorithms are merely used to demonstrate the processing power of the architecture. Other applications are for example high level image processing and matrix operations.

## 7. Acknowledgements.

The authors would like to thank L. Pieloor, A. Huijgen and W. Siteur for their contribution to this project.

## 8. References.

[1] Arvind, D. E. Culler, Dataflow Architectures, MIT / LCS / TM-294, Massachusetts, 1986.

[2] A. Huijgen, Design of a data-driven computer architecture for local neighborhood operations, Twente University, Enschede, internal report 88V022, march 1988.

[3] Product description μPD7281 and μPD9305, NEC Electronics Europe, Dusseldorf, 1986.