REAL-TIME EMBEDDED IMAGE PROCESSING APPLICATIONS USING THE A^3 METHODOLOGY

Y. Sorel

INRIA, Domaine de Voluceau, Rocquencourt
B.P.105 – 78153 Le Chesnay Cedex – FRANCE
Email: yves.sorel@inria.fr

ABSTRACT

In order to cope with the needs involved by real-time embedded image processing applications, multicomponent architectures are required. We used the well known Edge Detection algorithm to demonstrate how the A^3 methodology (Algorithm Architecture Adequation), and the CAD software SynDEx which supports it, may improve the implementation of such algorithms on a multi-DSP architectures. The proposed approach, based on accurate algorithm and hardware architecture models, reduces significantly the development cycle of image processing applications, by simplifying test and debug process. Moreover, it allows to minimize the hardware resources as required for embedding, while satisfying real-time constraints.

1. REAL-TIME EMBEDDED APPLICATIONS

Real-time embedded signal processing applications are basically ***reactive systems*** which must permanently interact with their environment that they control [1]. They react to input stimuli received from the environment, by triggering computations to generate output reactions and/or change their internal state. In order to keep control over their environment, they have to react within ***bounded duration***. When the ratio between the amount of computation involved by the application algorithm, and the reaction duration (response time) boundary is high, standard off the shelf sequential architectures are inadequate, and parallel ***multicomponent*** architectures are required. Programming such architectures is an order of magnitude harder than with mono-component sequential ones, and even more, when hardware resources must be minimized to match cost, power and volume constraints, imposed by embedded applications.

2. MULTICOMPONENT ARCHITECTURES

Although new mono-component uniprocessor architectures (workstations), provide ever increasing computation power, they cannot cope with the ever increasing complexity of some image processing applications. Parallel architectures are needed, as well to satisfy real-time constraints (computation load repartition), as to take into account the distributed nature of the resources (sensor/actuator, computation, memory) inherent to real-time embedded applications. This leads to multicomponent architectures, built from different types of programmed components (CISC or RISC processors, DSP, micro-controllers) and/or of specific components (dedicated boards, ASIC, FPGA), all together connected through a network, built from different types of communication media (point-to-point serial or parallel links, multi-point shared serial or parallel buses).

3. A^3 METHODOLOGY

A^3 means Algorithm Architecture Adequation. The goal of the methodology is to find out an ***optimized implementation*** of an application algorithm on an architecture, while satisfying constraints [2]. "Adequation" is a French word meaning an ***efficient*** matching. Note that it is different from the English word "adequacy" which involves a ***sufficient*** matching. A^3 is based on graphs models to exhibit both the ***potential parallelism*** of the algorithm and the ***available parallelism*** of the multicomponent. The implementation is formalized in terms of graphs transformations.

3.1. Algorithm Specification and Verification

The classical notion of algorithm, given for example by Turing [3], defining a finite total order on the execution of a finite multi-set of operations, is here extended to partial order. However, this partial order is different from the one obtained by Hoare's Communicating Sequential Processes approach [4]. The algorithm is modelized here by a ***conditioned data-flow graph***, that is, a folded dependency graph presenting a pattern indefinitely repeated. This dependency graph describes data-dependency relations (edges) between operations (vertice). A data-dependency is actually a data transfer from a producer operation to a consumer operation. This involves a partial order on the execution of the operations, called ***potential parallelism***. Potential means that this parallelism will be exploited only if parallel hardware resources are available. The execution of each graph pattern of the dependency graph is triggered when an input event, coming from the environment, is received by operations without predecessor. The output events are sent to the environment by operations without successor. Moreover, a dependency graph may be conditioned, that is, a part of this graph may not be executed. Specific conditioning vertice, with two input and one output, do not produce data when their conditioning input, which must be of

boolean type, carries a false value. In this case, by transitivity all the dependent vertice will not be executed. Otherwise, when both input are present, but the conditioning one is true, the output takes the value of the other input. At this point, it is assumed that the result produced by an operation is simultaneous with the input which triggers it. This allows to make verifications on this specification in terms of input output events ordering. Note carefully here, that operations and data transfers durations are not taken into account. It turns out a logical time, which instants correspond to the input output events interleaving. This algorithm model has the semantics of the synchronous languages and more specifically, of the synchronous data-flow language SIGNAL [5].

3.2. Multicomponent Specification

The implementation consists in *distributing* and *scheduling* the algorithm data-flow graph on the multicomponent taking into account real-time constraints. The operations and the data transfers durations are now considered, and therefore exploited to optimize the implementation, that is, to satisfy real-time constraints and to minimize the resources. In order to improve the usual approach based on a coarse model of the architecture like PRAM or DRAM [6], the multicomponent architecture is modelized more accurately. It is an hyper-graph which vertice are *components*, and which hyper-edges are *communication media*. A component may be in turn described in terms of an hyper-graph, leading to a hierarchical decomposition of the hardware architecture.

The smallest atomic component is a finite state machine. Two types of components are distinguished. An *operator* sequences operations on data, read from/written to communication media. Each *operation* of the algorithm graph is an indivisible sequence of computations. A *transformator* sequences data transfers between communication media (DMA, serial/parallel ...). Each transfer is an indivisible sequence of reads on one communication medium, interleaved with writes on another communication medium, in a ratio depending on the relative widths (number of parallel data wires) of the two communication media. A *medium* hyper-edge encompasses the wires used to support the communication, the finite state machine which arbitrates and synchronizes accesses to the wires, and some memory. Random access memory allows asynchronous communication, whereas sequential access memory (FIFO) implies synchronous communication. It is worthy to notice here, that it is very important during the optimization, to take into account carefully communications which support data transfers, especially if they are routed. Moreover, arbitrations must not be neglected because they introduce also delays. Figure 1 shows a multicomponent built from two TMS320C40, described with this model at the lowest level. Each processor is the sub-hyper-graph composed of a CPU operator and six CP transformators (the six comports associated to the six links of the TMS320C40) which share a unique DMA engine (hyper-edge DMA). The CPU and the comports communicate through a shared memory (hyper-edge MEM). The two processors communicate through three links (hyper-edge LINK1, LINK2, LINK3) which may operate in parallel.

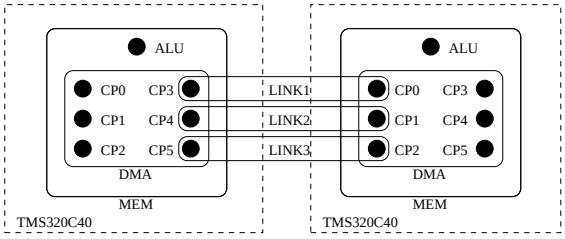

Figure 1: Bi-C40 Architecture

The architecture hyper-graph is labeled by characteristics related to hardware considerations. Because, afterwards in this paper, only the reaction duration will be considered for the optimization, we focus here on the durations. However, other characteristics could be taken into account, such as, amount of necessary memory, power consumption .... Then, to each operator is associated the list of couples (operation, duration) it is able to perform. Similarly, to each transformator is associated the list of couples (data transfer, duration) it is able to perform. Furthermore, when several components share a resource (sequencer, memory ...) arbitration is necessary. Consequently, the durations associated to the operations assigned to the involved operators must be modified according to an *interference matrix* which modelizes a slowing down. The table 1 describes a TMS320C40 characterization without interference, the left one is associated to the ALU operator, the right one is associated to the DMA transformator.

C40alu	
fft256	1250
mul10	14
add10	14

C40dma	
logical	6
integer	6
[10]real	50

Table 1: TMS320C40 Alu and DMA characterizations

The table 2 describes the interference matrix when an input and an output DMA transformators, associated to a TMS320C40 link, share the same link. When one is active the other one is slowed down in a 50 percent proportion. This matrix is used to weight the durations given above.

C40link	dmaI	dmaO
dmaI	-	50%
dmaO	50%	-

Table 2: TMS320C40 Link interference matrix

3.3. Adequation

The implementation is formalized in terms of transformations applied on the previously defined graphs. The distribution is a spatial allocation (assignation of several op-

erations resp. data transfers, to an operator resp. transformator). It induce inter-operator communications, possibly routed when operators are not directly connected. The scheduling is a temporal allocation (ordering through the operator resp transformator sequencer, of the assigned operations resp. data transfers). Given an algorithm graph and a characterized architecture graph, the optimization consists in selecting among all the possible transformations, on the one hand the one which maintains the input output event ordering verified during the algorithm specification, and on the other hand the one which minimizes the reaction duration and the number of components. This problem cannot be solved optimally (NP complete), the selection is carried out by fast ***resources allocation heuristics*** based on list scheduling greedy algorithms [7] which give as better results as they rely on an accurate architecture model.

From a selected graphs transformation, it is possible to generate automatically an optimized distributed executive for the programmed part of the architecture. It is built from a library of architecture-dependent executive primitives composing the ***executive kernel***. There is one executive kernel for each supported processor.

4. OPTIMIZED IMPLEMENTATION OF AN EDGE DETECTION ALGORITHM

We chose the well known Edge Detection algorithm [8] (average filtering, gradient using a four directions Kirsch mask convolution, binarization and thinning) to show how the SynDEx software, which supports the A^3 methodology, assists the user during the implementation of image processing algorithms, in satisfying real-time constraints, and minimizing hardware resources in order to meet embedding requirements.

4.1. Using SynDEx

SynDEx is a system level CAD software, its Graphical User Interface (GUI, see figure 2) allows the user to perform the following tasks. First the user specifies the algorithm as a data-flow graph to exhibit potential parallelism. A rather fine granularity of the parallelism is chosen at this initial step. Given an algorithm and a set of characterized components, SynDEx may propose a first architecture, which number or components is the first integer greater or equal to the ***maximum speed-up***. It is the ratio between the sum of all the operations durations and the critical path duration. Then, the user connects together the components specifying the architecture as an hyper-graph. Now, he may ask for an "adequation", involving the execution of SynDEx's heuristics. They propose a distribution and a scheduling, displayed as a predicted timing diagram. This one depicts the beginning date and the ending date of each operation (computations as well as communications) involved during the execution of the algorithm on the architecture. The timing diagram height corresponds to the duration elapsed between the input data event, and the output data event produced by the algorithm execution. This value corresponds to the predicted reaction duration. Considering it, the user checks if the real-time constraint is satisfied. If it is, he may attempt to decrease the number of components. If the real-time constraint is not satisfied, he may try to improve the result given by the heuristics. In order to do this, he may impose distribution constraints forcing certain operations to be assigned to particular components, and runs again the adequation. If this fails, lastly he must re-design the algorithm, generally he modifies first the granularity that he chose previously. When the user is satisfied with the predicted timing, he asks for the generation of a dead-lock free executive for the real-time execution of the algorithm on the multicomponent. On each processor, the executive is generated in an intermediate macro-code, independent of the processor instruction set. The set of macros composing the executive kernel provides the following primitives: operations of the algorithm graph (addition, multiplication, sliding windows ...and user specific macros), inter-component communications, initialization of program memories, execution in pseudo-parallelism of sequences of operations and inter-component communications, and real-time performances logging. SynDEx comes presently with executives kernels for: Transputer T80X, TMS320C40, i80-386, i8051. Executive kernels for other processors can be easily ported from the existing ones.

The Edge Detection algorithm was implemented on a multi-DSP architecture with SynDEx. The figure 2 shows a screen snapshot of the GUI. The left window displays the algorithm (lower left graph) and the architecture composed of four interconnected TMS320C40 DSP (upper left graph). The right window displays the resulting predicted timing diagram computed by the heuristics, for one input image processing. Time flows from top to bottom, in each processor column the operation height is proportional to its duration, and each inter-processor communication is represented by a line joining its source and destination. The height of the timing diagram corresponds to the reaction duration of the Edge Detection for one image.

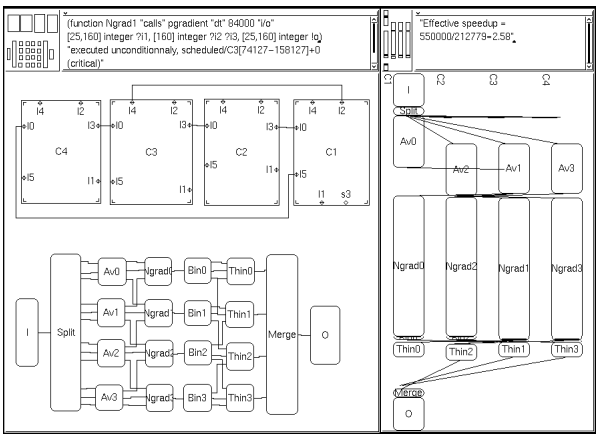

Figure 2: SynDEx Graphical User Interface

4.2. Parallelization and Granularity

As mentioned before, when it is time to implement an image processing algorithm running in real-time, a multicom-

ponent architecture is required. It turns out that the algorithm must be specified with potential parallelism. The data-flow graph model allows to describe the three kinds of potential parallelism: control parallelism (two operations without any data-dependency, might be executed on two different operators), data parallelism (several identical operations whose input receive different data, without any data-dependency, might be executed on several different operators), finally functional parallelism is inherent (two operations with a data-dependency might be executed in pipe-line). The most difficult issue when dealing with potential parallelism concerns the ***granularity***, that is to say the function complexity given to each atomic operation of the data-flow graph. For example, does an operation represent a simple multiplication or a dot product (several multiplications accumulations). Remember that a sequence of instructions is associated to each operation. Moreover, when data parallelism is considered the granularity relies also on the size of the data processed by each identical operation. To find out the best level of granularity, is always an hard task. This involves an iterative process taking into account constraints (available operators and transformators as well as real-time performances). The final solution is a compromise between granularity and hardware resources. SynDEx offers a software environment suited to find this compromise. As a matter of fact, it is very convenient with SynDEx to test different granularities for the same algorithm using the predicted timing diagrams. When models are accurate enough to trust in, it is tremendously simpler than to actually program and test the algorithm in real-time.

The results given in figure 2, were obtained after several iterations inside the SynDEx environment. The first step consisted in specifying the Edge Detection algorithm, exhibiting control and data parallelism at the finest granularity, with the basic logic and arithmetic operations (addition, multiplication, and, or ...) applied on each pixel of each input image. The Edge Detection algorithm is very simple mathematically speaking. The filtering and the gradient involve mainly the convolution of a matrix made of the considered pixel and its eight neighbors by a 3x3 mask matrix. This operation involves a lot of multiplications accumulations and consequently a lot of data-dependencies. The binarization and the thinning require tests rather than arithmetic operations. The resulting output images produced by the algorithm are verified on a mono-processor, in terms of numerical values and also in terms of ordering, in respect of the values and the order of the input images. Obviously, this method based on a fine granularity, generates too much inter-processor communications (due to data-dependencies) when the algorithm is implemented on a multiprocessor, reducing the benefit brought by parallelism. Therefore, the granularity was increased from basic operations to more complex operations, and similarly from pixel to part of image. Thus, the Edge Detection algorithm was decomposed in as many identical sub-algorithms as the input image was cut in parts. Each sub-algorithm is a chain, that is, a graph composed of the four complex data-dependent operations: average filtering, gradient, binarization and thinning (see figure 2). The basic operations applied to each pixel and its eight associated neighbors, were extended to each image part and its four associated borders, because each part must take into account pixels belonging to neighbor image parts.

Several cuts of the input image were experimented using the predicted timing diagram of SynDEx. Finally it turned out that, the Edge Detection algorithm decomposed in four identical sub-algorithms (see figure 2), corresponding to a four horizontal stripes cut of the input image, ended up to the best compromise approaching as close as possible real-time, while fully using the power provided by the four TMS320C40 DSPs at our disposal. Here are the actual real-time reaction duration of the Edge Detection for one image, logged with the executives generated by SynDEx: 1610 ms with one TMS320C40 DSP, 850 ms with two DSPs, 480 ms with four DSPs. These values are for 160x100 pixels image and do not take into account the image acquisition duration because an image is always ready to be processed, thanks to a frame buffer using a "ping-pong" memory.

5. CONCLUSION

This case study has shown the benefits of A^3 and SynDEx for distributed real-time embedded image processing applications. Thanks to the accurate predicted timing due to the multicomponent model, thanks to the heuristics and to the automatic executives generation, the user's time was not wasted in writing and debugging distributed code, but was rather spent in an efficient parallelization of the application algorithm. This dramatically reduces the development cycle of such applications.

6. REFERENCES

[1] D. Harel, A. Pnueli. *On the development of reactive systems.* In K. R. Apt, editor, Logics and Models of Concurrent Systems, Springer Verlag, New York, 1985.

[2] Y. Sorel. *Massively Parallel Computing Systems with Real Time Constraints The "Algorithm Architecture Adequation" Methodology*. Proc. of Massively Parallel Computing Systems Conference, Italy, 1994.

[3] A.M. Turing. *On computable numbers, with an application to the* Entscheindungs *problem.* Proc. London Math. Soc., 1936.

[4] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[5] P. Leguernic, M. Leborgne, T. Gautier, C. Lemaire. *Programming real-time applications with SIGNAL.* INRIA Research Report 1446, June, 1991.

[6] M. Cosnard, A. Ferreira. *On the real power of loosely coupled parallel architectures.* Parallel Processing Letters, Vol. 1, 2:103-112, 1991.

[7] C. Lavarenne, Y. Sorel. *Performance Optimization of Multiprocessor Real-Time Applications by Graphs Transformations.* Proc. of the PARCO93 conference, France, 1993.

[8] A. K. Jain. *Fundamentals of digital image processing.* Prentice Hall, 1989.