AUTOMATIC COARSE-GRAIN PARTITIONING AND AUTOMATIC CODE GENERATION FOR HETEROGENEOUS ARCHITECTURES

M. Raulet[1,2], *M. Babel*[1], *O. Déforges*[1], *J.F. Nezan*[1]

(1) CNRS UMR 6164 IETR / INSA Rennes
20 av des Buttes de Coësmes
(2) Mitsubishi ITE TCL
1 allée de beaulieu
35000 Rennes, France

Y. Sorel[3]

(3) INRIA Rocquencourt
OSTRE team
B.P. 105 78153 Le Chesnay Cedex
France

ABSTRACT

Real-time signal, image, and control applications have very important time constraints, involving the use of several powerful numerical calculation units. The aim of our work is to develop a fast and automatic prototyping process dedicated to parallel architectures made of both PC and several last generation Texas Instruments digital signal processors: TMS320C6X DSP. The process is based on SynDEx, a CAD software improving algorithm implementation onto multi-processor architectures by finding the best matching between an algorithm and an architecture. SynDEx kernels for automatic PC and DSP dedicated code generation have been developed with the new SynDEx repetition feature. A full coding application (LAR) illustrates the results.

1. INTRODUCTION

Although new mono-processor architectures (Personal Computer) provide ever-increasing computational power, they cannot cope with the ever-increasing complexity of some control, signal and image processing applications. Parallel architectures are needed to satisfy real-time constraints (computation load balancing), as well as to take into account the distributed nature of the resources (sensor/actuator, computation, memory) of real-time applications. These heterogeneous architectures, built from different types of programmed components (RISC, CISC, DSP processors) and/or of non-programmed components (ASIC, FPGA, full-custom integrated circuits), all together connected through a network of different types of communication components (point-to-point serial or parallel links, multipoint shared serial or parallel buses, with or without memory capacity), are called multi-components.

The goal of a prototyping methodology is to go from a high level description of the application to its implementation onto a target architecture. Performances of the process can be evaluated by different aspects:
- maximal independency with regards to the architecture,
- possibility to handle heterogeneous multi-component architectures,
- maximal automation during the process (partitioning, code generation),
- efficiency of the implementation both in terms of executive time and resource requirements.

This paper presents a prototyping methodology based on SynDEx which satisfies the previous criteria for multi-processor architectures. Although SynDEx is basically a CAD tool for partitioning and code generation, we demonstrate that SynDEx can also be directly used as the front-end of the process for functional check. Furthermore, we show that data parallelism can be easily highlighted at a high level of description for distributed implementation purposes, through the new repetition feature of SynDEx associated with the data flow graph formalism. The methodology is illustrated with a video coder (LAR) automatically implemented onto a PC+Multi C6x platform.

2. SYNDEX V6

SynDEx[1] is a free academic system level CAD software developed in INRIA Rocquencourt, France. It supports the AAA methodology (Adequation Algorithm Architecture) for distributed processing [1].

2.1. Adequation Algorithm Architecture (AAA)

A SynDEx application (fig.1) is made of an algorithm graph (operations that the application has to execute) which specifies the potential parallelism, and an architecture graph (multi-components target, i.e. a set of interconnected processors and specific integrated circuits), which specifies the physical parallelism.

"Adequation" means an efficient mapping. Performing an adequation consists in executing heuristics which seek for an optimized implementation of a given algorithm onto a given architecture.

[1] www-rocq.inria.fr/syndex/

An implementation (fig.1) consists in distributing (allocating parts of algorithm onto components) and scheduling (giving a total order for the operations distributed onto a component) the algorithm onto the architecture. The scheduling is performed off-line [2].

In the AAA methodology, an algorithm is specified as a data flow graph infinitely repeated. Each edge represents a data dependence relation between vertices, and the corresponding vertices n-uple is ordered (i.e. its first element is the producer vertex while the other ones are the consumer vertices).

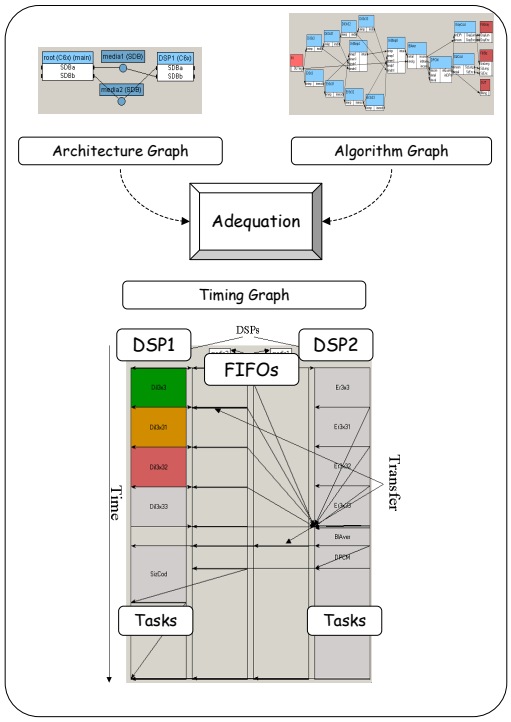

Fig. 1. SynDEx description graphs and the resulting timing diagram.

Moreover vertices are operations; operation stands for a sequence of instructions which starts when all its input data are available and which produces output data at the end of the sequence. In SynDEx, there is an additional notion of reference. Each reference corresponds to the definition of an algorithm (we will say "definition" instead of "algorithm definition" for the rest of this paper). The same definition may correspond to several references to this definition. An algorithm definition is a repeated data flow graph similar to those in AAA, except that vertices are references or ports. It enables hierarchical definitions of an algorithm. A reference in a definition may correspond to a definition which contains several references and so on. Ports are used only for propagating back and forth edges along the hierarchy.

2.2. Specification of regular algorithms

2.2.1. Principles

A regular algorithm typically presents "potential data parallelism" which is specified with a new feature of SynDEx allowing to repeat an algorithm. This is achieved by creating a reference to a definition such that the ratio between the number of elements of each input (resp. output) and the number of elements of the output (resp. input) it is connected to, is equal to the number of repetitions (Fig. 2). Some inputs may have ratio one, while the common ratio is greater than one. In this case the input is replicated as many times as the common ratio (Fig. 3).

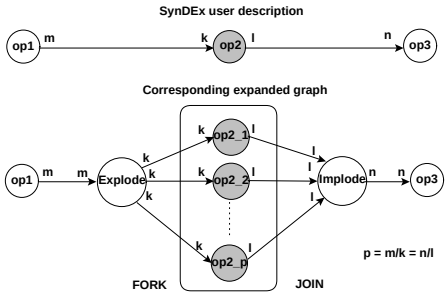

Fig. 2. Algorithm repetition.

Note that the number of repetitions applied to the definition is implicit. SynDEx determines this number from the algorithm specification and automatically repeat the algorithm definition and create the corresponding expanded graph (Fig. 2). The repetition factor is displayed next to the name of the reference. It is the expanded graph which will be taken into account during adequation and code generation. The expansion will introduce new vertices called "explode" and "implode". The explode vertex extracts each element of the data it receives for each repetition of the definition, whereas the implode vertex builds the data it sends by concatenating each separated element produced by each repetition of the definition.

2.2.2. Example

We illustrate the repetition principle used for specifying regular algorithm by the multiplication of a vector $V = \{v_1, v_2, \ldots, v_m\}$ by a scalar s giving a vector $W = \{w_1, w_2, \ldots, w_m\}$ as a result: $W = s \cdot V$.

A direct representation by a unique operation of a multiplication of a vector by a scalar does not express any parallelism. Nevertheless, data parallelism can be found at a finer granularity. Let m be the dimension of the vector. We may specify the repetition by m of a multiplication between two reals giving a real as a result, by referencing the definition of the multiplication and by connecting one of the two inputs to an output which is a vector of m real (V), and the

other input to an output of one real (s), and by connecting its output to an input which is a vector of m reals (W) (Fig. 3). The common ratio is m, and one input has a ratio one, that is to say the real is replicated m times in order to be multiplied by one of the m elements of the vector.

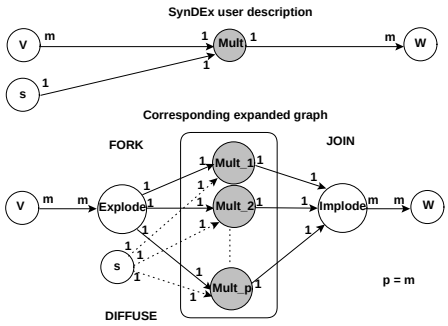

Fig. 3. Example of repetition: Multiplication of a vector and a scalar.

Such a description can be easily transcribed to image processing: data can be split into several parts (slices) that can be coded independently. So the process of these slices is parallelized through a similar scheme.

2.3. Automatic Executive Generation

The goal of SynDEx is to directly perform an implementation from an algorithm specification. It automatically generates a "generic executive" independent of the processor target into several source files (fig.4), one for each processor. Moreover, it generates automatically a makefile for automating the specific compilation chain of the architecture. These generic executives are dedicated to the application without any support of RTOS. They are composed of a list of macro-calls. The macro processor M4 transform this list of macro-calls into compilable code for the specific processor target. It replaces macro-calls by their definition given in the corresponding "executive kernel", which is dependent of the processor target.

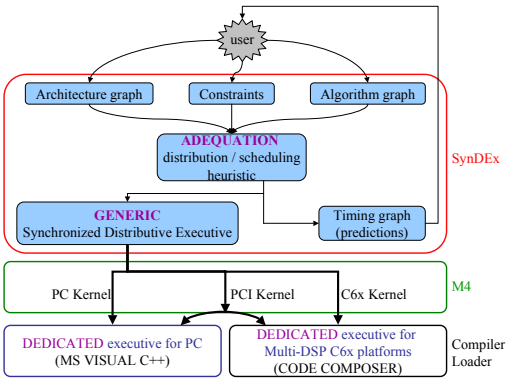

Fig. 4. SynDEx utilization global view

SynDEx kernels are available for the following processors: Analog Device ADSP 21060, SHARC, Motorola MPC 555 et MC 68332, Intel i80x86 et i8096, Unix/Linux workstations, Texas Instruments TMS320C40. The kernel for C6x was presented in detail in [3]. We had to develop a PC kernel and a PCI media interface in order to manage our heterogeneous platform described in the next section.

3. FAST PROTOTYPING METHODOLOGY

3.1. Platform

The chosen platform enables the user to obtain a coherent and modular target architecture. Our platform is composed of a Sundance motherboard and two SMT335 TIM modules. The SMT335 TIM consists of a Texas Instruments TMS320C6201 (64 KB of internal program memory, 64 KB of internal data memory) running at 200MHz. Modules are populated with 512KB of synchronous burst SRAM and 16MB of synchronous DRAM.

A Field Programmable Gate Array (FPGA) is used to manage global bus accesses and implement six communication ports (20 MB/s) and two Sundance Digital Buses (SDB). SDBs are 16-bit data parallel links achieving high-speed data transfers (200 MB/s each), an important point in regards with the large quantities of data (images and their related data) needed in video coding.

This platform is generally used in a *stand alone configuration*. Video algorithms can be characterized by a fix data driven scheduling [4], while requiring high performance processing. SynDEx does not need any RTOS like 3L-diamond product. It generates dedicated executives based on off-line scheduling [2] leading to the minimum overhead in space and time. Moreover SynDEx avoids inter-processor communication deadlocks.

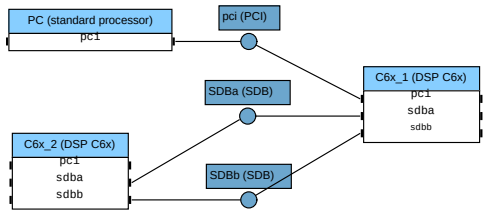

Fig. 5. SynDEx PC-multi-DSP architecture graph

The Sundance motherboard is a PCI card plugged in a PC. The description of this architecture is described with a graph in SynDEx (fig. 5). The motherboard PCI bus can be used in order to exchange data and to share processing with the PC. In this configuration, the multi-DSPs platform can then be seen as a *PC co-processor*. In terms of Real-Time Operating System (RTOS), the control part (file or network management for instance) can be supported by a preemptive on-line scheduling on the PC [2], while video algorithms

can be more efficiently implemented by a non-preemptive off-line one on the multi-DSPs PCI board.

3.2. Methodology

Our previous prototyping process integrated AVS as a front-end [3]. AVS is a software designed for algorithm data flow graph (DFG) specification and simulation. The DFG was validated thanks to AVS visualization tools and was automatically transformed to be compliant with SynDEx algorithm input.

The work presented here is based on the use of SynDEx for the specification and for the simulation of the algorithm graph without any commercial tool like AVS. Hence, SynDEx allows the full rapid prototyping starting from the application specification (DFG) to the final multiprocessor implementation (fig.4) in four steps:

Step 1: the digital image designer creates the DFG of his application with SynDEx. The automatic code generation provides a standard C code for a single PC implementation. In this way, the user can design each C function associated with each vertex of its DFG, and can check the functionalities of the complete application with standard compilation tools like Visual C++. The automatic code generation allows the use of visualization primitives (instead of AVS visualization tools) for an easy functional check of image and video algorithms.

Step 2: the DFG developed is then used for the automatic prototyping onto monoprocessor targets with chronometrical reports inserted automatically by the SynDEx code generator. The execution duration associated to each function (i.e vertex) executed on each processor of the architecture graph (PC and C6x) is automatically estimated through dedicated temporal primitives.

Step 3: the user can easily use these durations to characterize the algorithm graph by entering these values in SynDEx.

Step 4: SynDEx generates a real-time distributed and optimized executive, where chronometrical report are not inserted, according to the target platform. Several platform configurations can be simulated (processor type, their number, but also different media connections).

The main advantage of this prototyping process is its simplicity, because most of the tasks realized by the user concern the application specification with his usual compiling environment. The required knowledge of SynDEx and of compilers (Visual C++ and Code Composer) is limited to simple operations. All complex tasks (adequation, synchronization, data transfers and chronometrical reports) are executed automatically.

4. LAR IMAGE COMPRESSION METHOD

A new image compression algorithm has been developed in our laboratory: its implementation on a mixed architecture provides a validation of our fast prototyping methodology. This algorithm, called LAR (Locally Adaptive Resolution), is an efficient technique well-suited for image transmission via the Internet or for embedded systems. Basically, the LAR method was dedicated to grayscale still image compression, but extensions have been also proposed for colour images and videos, and lossless still image compression [5].

4.1. General principle of the method

The basic idea of the LAR method is that the local resolution (pixel size) can depend on the activity: when the luminance is locally uniform, the resolution can be low (large pixel size). When the activity is high the resolution has to be finer (smaller pixel size). A first coder is an original spatial technique and achieves a high compression ratio. It can be used as a stand-alone technique, or complemented with a second coder allowing to encode the error image from the first coder topology description. This second one - called a spectral coder - is based on an optimal block-size DCT-transform. This study concerns only the first spatial coder: figure 6 presents its global process.

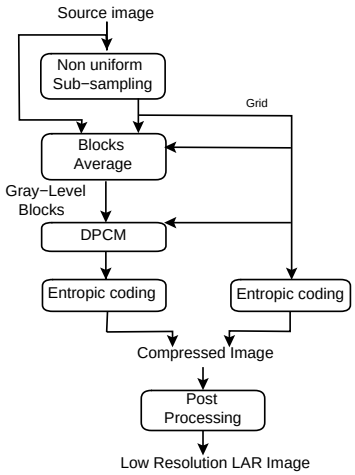

Fig. 6. Global scheme of the spatial coder.

4.2. Spatial coder

The image is first sub-sampled by 8×8 squares representing local trees. Then, each one is split according to a quad-tree scheme depending on the local activity (edge presence). The finest resolution is typically 2×2 squares. The image can be reconstructed by associating each square with the corresponding average luminance in the source image. The image contents information given through the square size is considered advantageous for the luminance quantization. Indeed, large squares require a fine quantization, as they are located in uniform area (strong sensitivity of human eye to brightness variations). As for the small ones, they support a coarse quantization as they are upon edges (low sensitivity).

The intensity value of each block is encoded through a DPCM approach. The process is done in one pass (raster scan), by means of the Gradient Adjusted Predictor.

Size and luminance are both encoded by an adaptive arithmetic entropic encoder. The average cost is less than 4 bits per square. Perceptible blocks artifacts in homogenous areas are easily removed by simple but efficient post-processing based on adaptive linear interpolation. Moreover, the previously described content-based decomposition leads to propose an automatic coarse edge-driven segmentation at both the coder and the decoder.

4.3. Construction of the grid

The size of the pixels is determined by the estimation of the local activity. For that purpose a morphological gradient is calculated inside a square area: if the difference between dilated and eroded values (maximal value - minimal value) is low, the square size can be increased. A straightforward implementation of this procedure requires a lot of computational time. A more suitable solution consists of decomposing the process into a succession of elementary operations.

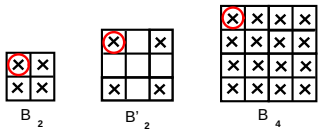

Fig. 7. Structuring elements.

Let B_2, B'_2 and B_4 (see figure 7) be structuring elements. One can easily show that

$$B_4 = B_2 \oplus B'_2. \tag{1}$$

If I denotes the image matrix, according to the Minkowski addition, the erosion and the dilation by B_4 are defined respectively by

$$\begin{aligned} I \ominus B_4 &= (I \ominus B_2) \ominus B'_2, \\ I \oplus B_4 &= (I \oplus B_2) \oplus B'_2. \end{aligned} \tag{2}$$

Originally the use of the structuring element B_4 requires 16 operations, whereas the Minkowski property implies now only 8 computations.

For our purpose, we only need to handle the odd elements of our image. Let $\downarrow$ denotes the sub-sampling operation, the equations 2 become

$$\begin{aligned} (I \ominus B_4) \downarrow 2 &= ((I \ominus B_2) \ominus B'_2) \downarrow 2, \\ (I \oplus B_4) \downarrow 2 &= ((I \oplus B_2) \oplus B'_2) \downarrow 2. \end{aligned} \tag{3}$$

As

$$B'_2 \downarrow 2 = B_2,$$

the equations 3 are equivalent to

$$\begin{aligned} (I \ominus B_4) \downarrow 2 &= ((I \ominus B_2) \downarrow 2) \ominus B_2, \\ (I \oplus B_4) \downarrow 2 &= ((I \oplus B_2) \downarrow 2) \oplus B_2. \end{aligned} \tag{4}$$

Figure 8 shows the different stages of our non-uniform decomposition, based on the previous described technique. We first compute the gradient inside 2×2 blocks and if the value obtained is lower than a given threshold, the resulting matrix is sub-sampled by two before repeating the morphological process.

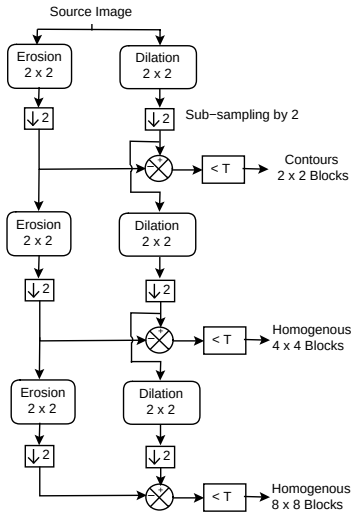

Fig. 8. Non-uniform decomposition

5. LAR CODER IMPLANTATIONS

5.1. SynDEx/LAR application

A first functional check of the still image LAR coder was realized on a PC. As the generated code is totally portable on DSPs (programs in C), this coder has been automatically implemented onto multi-processors (PC + multi-DSPs). There is no noticeable acceleration between one and two DSPs: this is due to the fact that the algorithm was not specified with enough potential parallelism. This is why we increased the potential parallelism by splitting the image into two slices (two is equal to the number of available processors). Then, the coder is repeated twice (fig. 2) presenting data parallelism.

5.2. Distributed implementation results

As we intend to develop a video application, the processing time is critical and must be exactly equal to 40ms (video constraint). Here we lay the emphasis on CIF images (Mpeg4 image format: 352*288 pixels).

We simulate $20ms$ per CIF image on the PC which is not an embedded system. Our embedded system is composed of two C6201 DSPs which perform all the coder processing. Once the functional description is checked on the PC, the role of the PC is to send the data corresponding to the image to be coded and then to receive the data from the platform in order to display the reconstructed image. The

PC and PCI media have been added on the SynDEx graph, in order to get an automatic code generation. $200ms$ are needed to encode an image with one DSP, and $120ms$ with two DPSs. The gain factor is 1.6. Maximum rate could be 2. Indeed the available internal memory size of the C6201 is not sufficient to store all the data, that is why the processing time is here so large and does not respect the real-time processing. All the data required for our coder is allocated in the external memory which lets up our application. Some simulation on a short part of the CIF image (16 lines) shows us that we could expect an accelerating factor of 7.2. Our DSPs memories are undersized for our application. Future developments will integrate new C6416 DSP running at 400Mhz - twice as fast as C6201- and providing 1MB of internal memory - fifteen times more than C6201's memory - which is enough to store all the data needed to the coder. So we could obtain an accelerating factor of 14.4 and could compute the CIF image in $8.5ms$ with two C6416 and we could consequently consider our platform as an efficient co-processor.

6. CONCLUSIONS AND PERSPECTIVES

This paper has presented a distributed LAR encoder implementation along with its design methodology, allowing both an automatic distributed implementation and a minimal embedded code. The multiprocessor implementation is optimal thanks to SynDEx and more particularly its repetition feature. This feature allows to describe the split of an image into slices in the SynDEx algorithm graph, increasing the potential parallelism of the application.

This paper also describes visualization primitives that enable the use of SynDEx as the front-end of the prototyping process instead of AVS. The main advantage is to decrease the complexity of the complete process. Another advantage is that SynDEx can be freely downloaded whereas AVS is a commercial tool. Algorithm mapping, synchronizations and data communications are automatically generated and optimized for the PC-multi-DSP target architecture.

The implementation of elementary and regular operations (DCT for instance) onto a non programmable component such as FPGA would give higher performances. We have work in progress in order to study co-design approach with SynDEx [6] for the automatic implementation of these regular operations onto a non programmable component.

We are also working on an Mpeg-4 coding application. As the application described in this paper, an Mpeg-4 codec can be fully parallelized using slices so that multi-components architectures will be used to reach real-time performances.

This work is partially supported by Mitsubishi Electric ITE-TCL. A SynDEx description of an UMTS application [7] is actually developed by Mitsubishi and will take benefits of the results presented here.

7. COPYRIGHT FORMS

8. REFERENCES

[1] T. Grandpierre, C. Lavarenne, and Y. Sorel, "Optimized rapid prototyping for real time embedded heterogeneous multi-processors," *7th International workshop on Hard-ware/Software Co-Design, IEEE Computer Society, ACM SIGSOFT, IFIP*, pp. 74–78, May 1999.

[2] F. Balarin et al., "Scheduling for Embedded Real-Time Systems," *IEEE Design and Test of Computers*, January-March 1998.

[3] J.F. Nezan, M. Raulet, and O. Deforges, "Integration of mpeg-4 video tools onto multi-dsp architectures using avsyndex fast prototyping methodology," in *IEEE Workshop on Signal Processing Systems (SIPS)*, October 2002.

[4] J.F. Nezan, *Integration de services video Mpeg sur architectures paralleles*, Ph.D. thesis, IETR INSA Rennes, November 2002.

[5] M. Babel, O. Déforges, and J. Ronsin, "Lossless and Lossy Minimal Redundancy Pyramidal Decomposition for Scalable Image Compression Technique," in *Proceeding of IEEE ICASSP 03*, Hong Kong, April 2003.

[6] S. Le Nours, F. Nouvel, and J.F. Helard, "Example of a co-design approach for a mc-cdma transmission system implementation," *Journees Francophones Adequation Algorithme Architecture (JFAAA)*, December 2002.

[7] C. Moy, A. Kountouris, L. Rambaud, and P. LeCorre, "Full digital if umts transceiver for future software radio systems," in *Proceeding of ERSA'01*, Las Vegas, June 2001.