Generic Programming Methods for the Real Time Implementation of a MRF Based Motion Detection Algorithm on a multi-processor DSP with multidimensional DMA

Frantz LOHIER[1,2], Lionel LACASSAGNE[1,2], Pr. Patrick GARDA[2]

[1]Electronique, Informatique, Applications (EIA)
Burospace, Bat. 4, Route de Gisy, 91571 Bièvres Cedex, France

[2]Laboratoire des Instruments et Systèmes (LIS)

Université Pierre et Marie Curie (UPMC), 4 place Jussieu – B.C. 252, 72252 Paris Cedex 05, France

frantz@lohier.com, lionel@lis.jussieu.fr, garda@lis.jussieu.fr

Résumé – Cette communication adresse une double problématique. D'abord, nous soulignons le besoin de méthodes de programmation génériques pour l'implémentation temps réel (TR) d'algorithmes de traitement d'image bas niveau complexes sur des architectures DSPs parallèles à base de multiprocesseurs exploitant le parallélisme au niveau instructions et de DMAs multidimensionnels. Ensuite, nous introduisons le besoin d'une implémentation TR d'un algorithme de détection de mouvement sur des architectures compatibles avec des systèmes bas coût embarqués. Pour répondre à ces besoins, nous montrons comment une méthodologie de gestion des flots synchrones reposant sur le DMA et qui se veut dynamique et générique sur le plan des configurations de traitement (suivant la nature des chaînes de traitement, la taille des images et du nombre de processeurs impliqués) peut être utilisée pour l'implémentation d'une méthode Markovienne de détection de mouvement sur l'architecture parallèle avancée du TMS320C80. Cette étude de cas montre l'adéquation de notre méthode et introduit un facteur d'accélération de 4 par rapport aux durées de traitement précédemment publiées de l'algorithme ciblé. Plus encore, on estime que le traitement TR est possible sur des images 256^2 avec un système C80 optimal.

Abstract – This paper addresses 2 problems. First, we emphasize the need for generic programming methods for the real time (RT) implementation of complex low level image processing algorithms on parallel DSPs featuring multi-processing and ILP (Instruction Level Parallelism) and multidimensional DMA. Second, we show the need for a RT implementation of a motion detection algorithm on hardware platforms suitable for low cost embedded systems. To tackle these issues, we show how a DMA based SDF (Synchronous Data Flow) methodology that is dynamic and generic in terms of processing configurations (according to the processing chains, image sizes and number of processors involved) can be used to implement a MRF (Markov Random Field) based motion detection algorithm on an advanced parallel DSP architecture: the TMS320C80. This case study shows the adequacy of our approach and demonstrate a speed factor of 4 compared to previously published implementations for the targeted algorithm. Furthermore, we estimate that RT performance can be achieved for 256^2 images on an optimal C80-based system.

1. Introduction

1.1 A novel approach to optimize 2D I/O streams

While analyzing the recent architectural evolutions, it appears that on the one hand :

- RISC and DSP cores are converging towards combined VLIW/SIMD architectures ;
- DMA co-processors now stand as the only major architectural difference between the RISC and DSP world. They offer more predictability at the cost of a more difficult programming stage (they feature self-modifying parameters and multi-dimensions support as in [4]).

Current programming methodological trends, on the other hand, are still limited :

- DSP development tools are lacking whereas implementation is raising in complexity (SIMD/software pipelining, C8x's VLIW operations, advanced DMAs) ;
- DSP flexible simulation/implementation platform (such as MatLab, Ptolemy or Hyperception) are not suited for 2D processing. For these applications, the 1D synchronous-data-flow (SDF) representation domain seems the most mature ;
- The code generated from these platforms isn't optimal and adding new processing kernels is often lacking a flexible re-usable framework.

To tackle these issues, we recently introduced a SDF oriented programming methodology [2][3]. It seeks the enhancement of data locality by chaining the execution of processing operators (i.e nodes) hence minimising the global amount of parallel DMA transfers. This approach grounds on "templates" which set up a generic framework to expand node's libraries and compose complex processing chains dynamically. Templates derive from the structuring element required/produced by algorithmic operators in the sense that they also gather node's implementation constraints. These additional multiplicity constraints relate to optimisations techniques such as the use of SIMD operation, loop unrolling or software pipelining [1][3]. These techniques maximise the usage of processor's resources through the execution of several algorithmic loop iterations in parallel. In a multi-processor context, the ultimate goal of our approach is to set up all the synchronous DMA requests from a generic chain's description, enhancing data locality while maintaining instruction caching low.

Based on this methodology we have implemented a complete low-level image processing library (more than 60 nodes) on the C80. This library features dynamic parsing of chains' description (nodes' template, image sizes, data cache sizes, number of processors), automatic DMA requests generation and synchronisation with processing. The design space isn't automatically explored towards an optimal use of heterogeneous computational resources and communication channels as with the Syndex' generic approach. Instead, partitioning on homogenous resource is user-guided but re-configurable at run-time and since it relies on optimized 2D I/O streams (in terms of data locality enhancement combined with advanced optimizing implementation techniques), it improves the parallelization efficiency.

1.2 The motion detection algorithm

To illustrate the use of our library, we describe the implementation of the robust Markov Random Fields (MRF) based motion detection algorithm proposed in [6]. Starting from an image difference, the ICM (Iterated Conditional Mode algorithm) is used to compute the resulting binary "label" picture in an iterative manner. It acts as a low-pass motion filter to locally minimize a spatio-temporal energy model. Iterations are performed at pixel level ("site recursive") and the suggested number of iterations is 4. Although this algorithm is sub-optimal, it nowadays involves a challenging computational load when seeking real-time (RT) processing on large images. Also, for this algorithm, the RT performance stand as an important criterion towards a good detection efficiency (especially when fast motion constitute the sequence).

A complete description of the approach is detailed in [2]. As a synopsis of it, a pre-possessing phase merges the absolute difference of 2 images, the variance of it and the binary thresholding of the absolute data (the initial labels E_{t+1}). This phase is followed by the ICM regularizing algorithm as shown in figure 1.

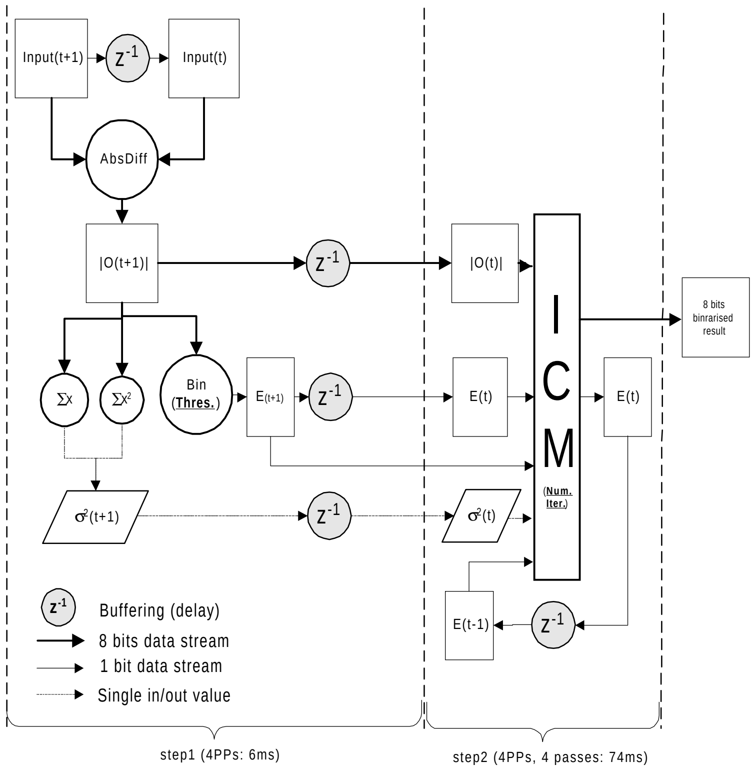

FIG. 1 : Synopsis of the MRF algorithm

2. The architectural mapping

The C8x has 1 general purpose RISC processor (MP) and 4 advanced SIMD/VLIW DSPs (or PPs for Parallel Processors) [4]. This suites the processor-farm model where the MP synchronizes the PPs representing the bulk processing power. The pre-processing and the ICM passes require just 2 image scans. For each scan, we use an SPMD partitioning scheme. The image is split among the 4 PPs and since each sub-region doesn't fit in the PP's general purpose internal data memory, the data are brought using multiple DMA requests while double buffering.

2.1 Implementing the pre-processing phase

The pre-processing stage is done in a single pass thanks to a chain connecting several nodes through their corresponding templates. Templates' geometry are mostly described by 4 parameters. $w_t \times h_t$ corresponds to the minimum amount of data that must be present in internal memory.

This quantity often exceeds the surface of the operators' structuring element because we integrate size requirements that raise while implementing generic optimization techniques towards the efficient algorithmical mapping onto hardware resources. Here, the goal is to lower the development burden and the granularity of each node by assuming we have a minimum amount of data that matches the number of parallel iterations processed in 1 cycle of the

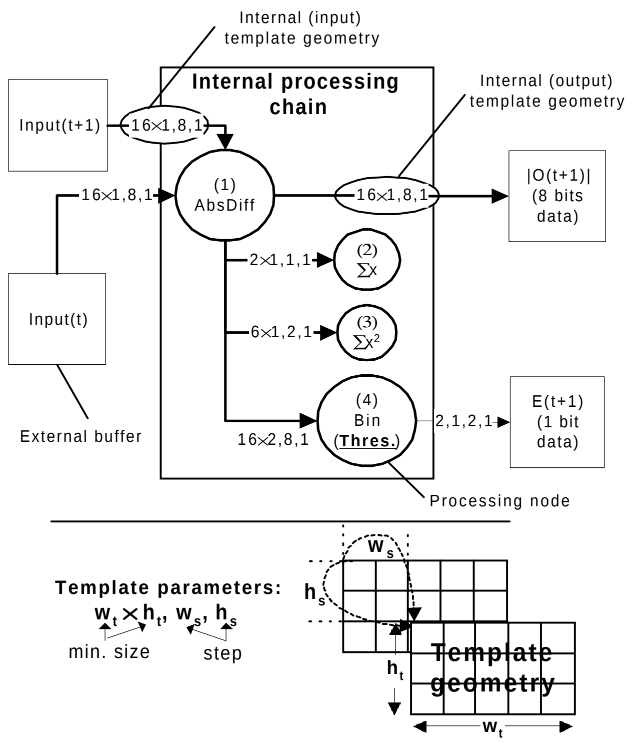

FIG. 2 : The pre-processing chain

nodes' loop kernel. Following the same goals, we also impose multiplicity constraints on the number of additional data that can be present in internal memory for the horizontal and vertical direction and described thanks to parameters w_s and h_s respectively. These templates allow the composing of complex processing chains which, together with the reduced granularity of nodes, allows to enhance data locality and reduce parallel DMA transfer and instruction caching towards improved real-time performance. Each node has synchronized input and output templates whose parameters are detailed on figure 2.

that connects to an external buffer. Propagation is based on template parameters and the solving of simple and independent diophantine equations (for the vert. and horz. direction). These equations appear as we try to synchronize the amount of data produced by a node with the number of data consumed by the next node in the chain. This synchronization mechanism, which involve nodes' templates, is depicted with figure 3. Recursively, the gained synchronization constants (which appear has 2 congruences: $\beta^o \equiv \phi(\Phi)$ et $\gamma^o \equiv \tau(\Gamma)$) are propagated between the output and input of each node until the leading node of each path is reached. This procedure enables the merging of the called "virtual-template" which features new parameters gathering all the size constraints ($w_t' \times h_t', w_s', h_s'$). Then, according to the original image size, we estimate the combined number of horizontal (β) and vertical (γ) virtual templates we can cache in an internal memory space of S-w_t'xh_t' bytes. A complete description of this procedure can be found in [2].

As a synopsis of our approach, table 1 shows (β,γ) couples we gain while applying this procedure on the first step of the motion detection algorithm with W=H=256 and S=2048 for all the input and output buffers. Min(β) and Min(γ) are then used to synchronize all the nodes as well as to partition data among all the processors (in a SPMD fashion). The subsequent transparent and dynamic programming of DMA requests allows to synchronize all input and output transfers (for all the processors) towards software managed data caches.

TAB. 1 : Generating & synchronizing DMA requests

Paths	Virtual template (w'_t, w'_s,h'_t,h'_s)	β	γ
I←AbsDiff←BinThres	16,8,**2**,1	**30**	**7**
I←AbsDiff←Σx	16,8,1,1	30	8
I←AbsDiff←Σx²	16,8,1,1	30	8
|O|←BinThres←AbsDiff	2,1,2,1	254	7

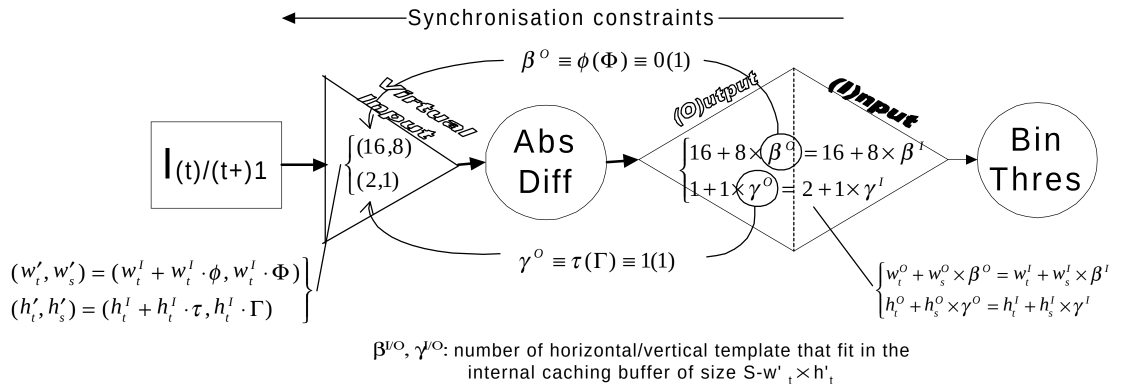
Synchronisation constraints
$\beta^O \equiv \phi(\Phi) \equiv 0(1)$
Virtual input
(O)utput
(I)nput
I(t)/(t+1)1
(16,8) (2,1)
Abs Diff
$16+8\times\beta^O = 16+8\times\beta^I$
$1+1\times\gamma^O = 2+1\times\gamma^I$
Bin Thres
$\gamma^O \equiv \tau(\Gamma) \equiv 1(1)$
$(w'_t, w'_s) = (w^I_t + w^I_t \cdot \phi, w^I_t \cdot \Phi)$
$(h'_t, h'_s) = (h^I_t + h^I_t \cdot \tau, h^I_t \cdot \Gamma)$
$w^O_t + w^O_s \times \beta^O = w^I_t + w^I_s \times \beta^I$
$h^O_t + h^O_s \times \gamma^O = h^I_t + h^I_s \times \gamma^I$
$\beta^{I/O}$, $\gamma^{I/O}$: number of horizontal/vertical template that fit in the internal caching buffer of size S-$w'_t \times h'_t$

FIG 3. Getting the virtual template

Next and since the processed images do not fit in internal memory, we need to estimate the number of all input and output parallel DMA requests between each synchronized execution of the chain based on the exact surface of data we can process. This calculus is based on the overall size constraints that are propagated along each path of the chain

Multidimensional support allows the DMA to address any region of interest and its arithmetic capabilities permits to implement the double buffering scheme with minimum processors' involvement. This greatly favors instruction cache coherency and hence, performance.

Thanks to this methodology and the use of generic optimization techniques (software pipelining, SIMD operations) to implement the nodes in assembly language, the pre-processing phase achieves 6 ms for a 256^2 image with 4 PPs working in parallel on a 40 Mhz device.

2.2 Implementing the ICM node

Thanks to the dynamic infrastructure of our C80 library implementing the generic DMA caching methodology, the system is reconfigured at runtime to run the ICM. This step is implemented as a single-node chain. The same I/O generating/optimizing algorithm is used but not detailed here. Instead, the various involved templates are shown in Figure 4.

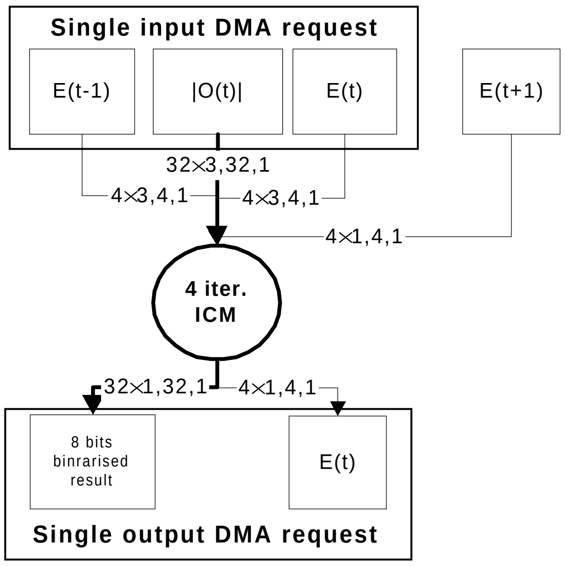

FIG. 4 : the ICM single-node chain

To lower the number of DMA requests required, some external buffer's data are organized sequentially and defines the ICM node as diadic. Also, there is no synchronization regarding the processed data that are shared between 2 processors. This simplification has very little impact on the motion detection efficiency and permits SPMD partitioning with maximum performance.

The ICM node is written in VLIW algebraic assembly language. We took advantage of the new code compactor (*ppca*) detailed in [5] to automatically gather assembly operations into 64 bits VLIW instructions. Register allocation was also done by *ppca* that efficiently compacted 313 operations into 199 VLIW instructions. Since the core of the node requires more register resources than available (each PP features 8 data registers and 2 sets of 5 address and 3 index registers), we manually inserted register spilling operations based on *ppca* feedback logs. We introduced the use of 3 LUTs (initialised by the MP) to fasten calculation whereas the site recursive version of the ICM algorithm prevented us from using the SIMD capabilities of the ALU (but this version of the ICM demonstrates faster converging).

The core of the kernel requires 40 VLIW instructions per pixel which is 4 times faster than what PP's optimising compiler produces on a C version of the same algorithm. On 256×256 images, at 40 Mhz, we measured 74 ms for the 4 ICM passes which is very close to the optimistic optimum: 256×256×40×4(num. pass)/(40Mhz×4(num. PP))= 65 ms.

3. Conclusions

This paper introduces two important results. Qualitatively, we demonstrate that our methodology is generic enough to cope with a complex low-level image algorithm. It is also flexible enough to be dynamic and independent of the image size and number of processors. Most importantly, with respect to generic optimisation techniques, it optimises 2D I/O streams and allows good speed-up towards RT performance.

Quantitatively and thanks to the framework of our general c80 image processing library, we gain a speed factor of 4 with respect to previously published processing duration for the same algorithm [6]. When considering the 60 Mhz version of the device, we can increase the processing rate up to 18 images per second for 256^2 images. The use of more efficient memory, like synchronous dynamic RAM (we used DRAM), would further increase the rate.

Pre-prop. & ICM	LIS-UPMC/EIA	LIS-INPG
128x128	-	DSP 96002: 15 images/s Cnaps 256 PEs: 10 images/s
256x256	40 Mhz C80: 12 images/s	-

Références

[1] *F. Lohier, L. Lacassagne, Pr. P. Garda*, Programming Technique for Real Time Software Implementation of Optimal Edge Detectors. Proc. of DSPWORLD'98
[2] *F. Lohier, L. Lacassagne, Pr. P. Garda*, A Generic Methodology for the Software Managing of Caches in Multi-Processors DSP Architectures. Int. Conference on Acoustics, Speech and Signal Processing. ICASSP'99.
[3] Articles [1] and [2] are available online at www.lohier.com
[4] TMS320C8X System-Level Synopsis, Texas Instruments SPRU113b.
[5] *Jihong Kim, Graham Short*, Performance Evaluation of Register Allocator for the Advanced DSP of TMS320C80. Proc. of ICASSP'98
[6] *A. Caplier, F. Luthon, C. Dumontier*, Real time Implementations of an MRF-based Motion Detection Algorithm. Journal of Real Time Imaging 1997.