ASYNCHRONOUS MULTI-CORE ARCHITECTURE FOR LEVEL SET METHODS

Eva Dejnožková and Petr Dokládal

School of Mines of Paris, Center of Mathematical Morphology, 35, Rue Saint Honoré,
77305 Fontainebleau, France, e-mail:{dokladal,dejnozke}@cmm.ensmp.fr

ABSTRACT

This paper proposes an asynchronous multi-core architecture for embedded systems using partial differential equations-based image processing algorithms. The study of data flow and the timing analysis is carried out in order to reveal optimal global architecture specifications. The global architecture uses a semi-parallel approach with several processing units running in parallel and shared memory blocks.

The results are illustrated by the implementation of a continuous watershed transform, followed by a discussion of the measured execution time and the computational load to demonstrate the efficiency.

1. INTRODUCTION

The Partial Differential Equations (PDE) based methods become popular because they offer an independence on the discretization grid, a better mathematical modelisation and a sub-pixel precision. The nowaday applications range from low-level (such as smoothing, denoising) to high-level image processing (such as segmentation by active contours or watershed) used for either static images, or sequences for object tracking. Other examples include recently proposed segmentation algorithms as the depth-, area- and volume-controlled continuous watershed [1], combinations of the level set methods with other techniques Principal Component Analysis (PCA) [2]. An overview of image improvement (PDE-based) operators can be found in [3].

In general, the application requires to solve non-linear PDEs. The numerical solution is obtained by recursive algorithms characterized by a high number of iterations. Another difficulty comes out from the use of hierarchical real-weighted data structures for some algorithms. Many authors have oriented their research to limite the iteration number by accelerating the convergence [4], by reformulating the sequential algorithm in a parallel way [5] or by eliminating the need of ordered data structures [6].

However, embedded systems remain rare. They are principally limited to the implementation of PDE-based filters [7]. As an example of specialized hardware for the active contours segmentation, one can cite implementation on a graphics hardware [8]. The objective of this paper is to define an embedded system architecture fitting the needs of the above-mentioned applications yet remaining sufficiently general and easily programmable with usual development tools.

First, we present an overview of the most frequent types of PDE-based algorithms. Then we concentrate on the analysis and optimization of the data-flow path to find an optimal load balance of all blocks of the architecture. Finally we propose a multi-core architecture built around several asynchronously operating RISC cores on one chip.

2. ALGORITHMS OVERVIEW

Typically, the result is obtained by deforming a given curve (propagation front) or surface with a given PDE. We consider only the Level Set formulation of the PDE-based methods [9] which can be classified as follows (u is the evolving function with u_0 as initial conditions):

1) **Surface propagation.** It includes diffusion filters, geometric smoothing, denoising, morphological operators with evolution controlled by equation: $\frac{\partial u}{\partial t} = \mathcal{F}(u)|\nabla u|$. In every iteration are processed all points in the image are processed. The temporal evolution is based on the local neighbourhood and generates the evolution of the level sets in the space [9]. The evolution stops as soon as the convergence or a given number of iterations is reached.

2) **Wave propagation.** The algorithms including weighted distance, continous watershed, Voronoï tesselations, Shape from shading are controlled by the Eikonal equation $|\nabla u| = \mathcal{F}$. The solution is propagated from the given sources on the entire image according to the speed $\mathcal{F}$ defined. The algorithm operates locally, only on the narrow band around the wave front. Typically, the front is propagated as equidistant to the sources by using ordered data structures [10].

3) **Deformable models.** We distinguish the types with regularizers (controlled by statistical information of regions), without regularizers (information of regions is not used) or combined with PCA. The general evolution equation has the form: $\frac{\partial u}{\partial t} = \mathcal{F}_{curvature}(u) + \mathcal{F}_{grad}(u) + \mathcal{F}_{region}(u)$. The algorithms proceeds by deforming a given initial contour. The deformation is controlled by a force obtained at each itera-

tion from the contour and geometrical (curvature, gradient) or statistical characteristics of the image (region intensity mean value) [9].

4) **Optical flow :** $\frac{\partial u}{\partial t} = f(\nabla u, I_1) + g(\frac{\partial I_2}{\partial x}, h)$, $\frac{\partial v}{\partial t} = f(\nabla v, I_1) + g(\frac{\partial I_2}{\partial y}, h)$. The motion vector is obtained by solving the systems of the above equations at each point in the image (I_1, I_2 are the successive sequence images, h is the searched motion vector field) [11].

If the image is considered as a continuous signal then the PDEs evolution can be seen as an iteration of a local filter operating on the neighbourhood [9]. From the implementation point of view, when updating a given point, only the local neighbourhood information is needed. Hence we have an important possibility of parallelism. The definition of the neighbourhood depends on the numerical scheme used for PDE discretisation (for details, see e.g. [9]). Below we assume the 4-neighbourhood and a more frequently used explicite Euler scheme for the integration. In order to analyse the data path, we organize the algorithms in two major groups (see Fig. 1): 1) **Global Scope**: algorithms that in one iteration process all points in the image and 2) **Narrow Band**: algorithms that in one iteration process only points close to the contour.

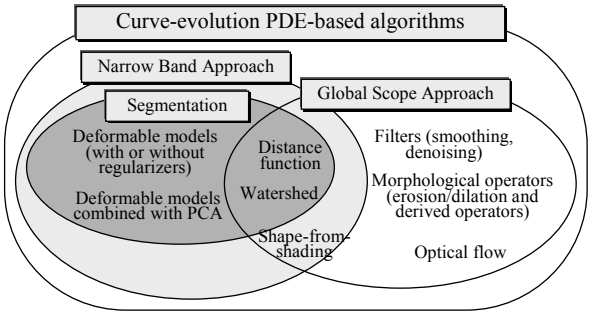


Fig. 1. PDE-based algorithms overview.

One iteration of both algorithm types can be written in the following form:

Algo. 1. General PDE algorithm

for all $p_i \in \mathcal{A}$ **do** *(in parallel)*
{ Retrieve_Neighborhood $u^n(N(p_i))$ and $u^n(p_i)$;
Calculate_Value $u^{n+1}(p_i)$;
Update_Value $u^{n+1}(p_i)$;
Activate_New_Points (insertion in $\mathcal{A}$); }

The set $\mathcal{A}$ contains points to be processed in one iteration. The Narrow Band type algorithms act only on the active points, i.e. points close to the current position of the travelling interface. Then $\mathcal{A} = \{p \mid |\operatorname{dist}(p)| < NB_{\text{width}}/2\}$, where NB_{width} is the width of the narrow band around the contour. The Global-Scope type algorithms act in every iteration on the entire image. Hence, each point in the image is active and the set $\mathcal{A} = \operatorname{supp}(I)$, $I =$image.

2.1. Data flow analysis

Inside one iteration, the new values of points are independent each of the other, hence the computation can be parallelized. Fig. 2 shows the data flow corresponding to the Algo. 1. The basic modules are the following ones (see Fig. 2): 1)**Processing units (PU):** compute the values of points according to the numerical scheme with the required accuracy. Several PUs operate in parallel. 2)**Data memory:** contains the images or other necessary informations as flags. 3)**Active points memory:** is used to store the active points in $\mathcal{A}$ (for the Narrow-Band-type algorithms).

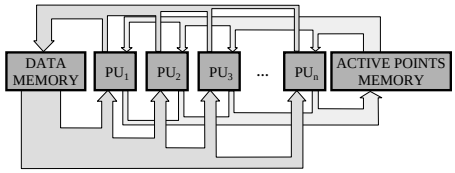


Fig. 2. Data-flow chart. The width of the paths corresponds to the volumes of data transitting on.

The outgoing data flow from the DATA MEMORY block is higher than the incoming data flow; indeed, to process a point p_i, a given PU has to extract its complete neighbourhood but it updates only one value. Thus, for 4-neighbourhood, the outgoing data flow is five times higher (the point p_i and its four neighbours) than the incoming one (updated point p_i). Similarly, at one moment, one PU can only read one point from the active points memory and may activate from one to several neighbours. Nevertheless, according to our experimental measurements, one processed point usually inserts one or two points (up to 90%). Therefore, the mean incoming data flow in the active points memory is only slightly higher than the outgoing data flow.

2.2. Timing analysis

The parallel running of the PUs on Fig. 2 generates simultaneous accesses to the shared memory. The optimal management can be obtained by considering also the execution timing. As results from Fig. 3, the code structure has two major

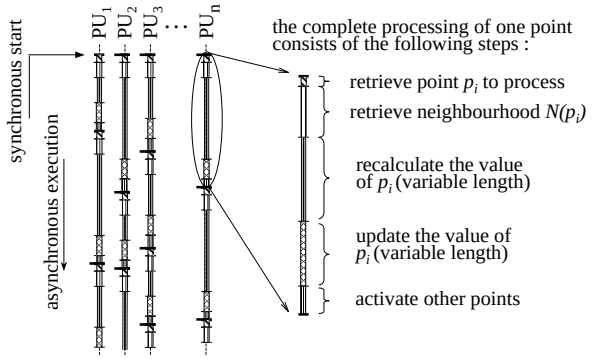


Fig. 3. Timing.

features: i) the retrieval of the point and of its neighbours

values is significantly faster than the recalculation, and ii) the recalculations of different points have different lengths because of *if*-conditions in the code.

Hence, the efficient execution of the code on several PUs has to be asynchronous. Recall, that asynchronous execution is possible because the point values are independent of each other. Moreover, asynchronous execution is advantageous because it makes the accesses random in time reducing the number of simultaneous memory manipulations.

3. GLOBAL ARCHITECTURE

As results from the above-given algotrithm analysis, the architecture must consider both memory random acces (Narrow Band type) and the entire image scanning (Global Scope type). To obtain a balanced activity of all the processing units we have adopted the semi-parallel approach where the data memory and the active points memory are shared (see Fig. 4). The Labels and Flags are used by the programmer for additional algorithm control and region propagation. The computation of $\mathcal{F}$, which is generally a non-linear function, represents the second challenge of an efficicent implementation. It seems necessary to use an ALU (Arithmetic Logic Unit). Therefore the choice of several parallely operating RISC cores with the optimized peripheries is straightforward. In our case, we have used four RISC processors with a usual instruction set. (The number of PUs is done by embedded cores available on existing FPGAs.)

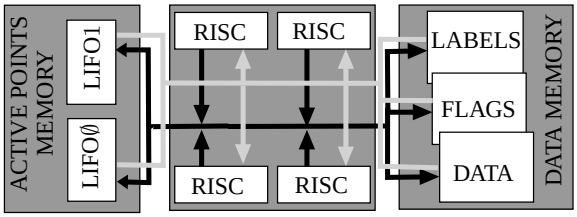


Fig. 4. Global architecture

To minimize the bus occupation the buses for reading and writing are separated. The same code is executed by each PU in an asynchronous way. Simultaneous accesses are handled by using semaphores. Whenever a simultaneous access to some memory occurs then the first arrived PU is served and the other is waiting until the memory is released. This principle simplifies the design of the shared blocks, and multiple-port blocks are not needed.

The active points memory contains the coordinates of the points to be processed. This memory is separated in two blocks. The currently processed points are in the one block and the the points activated for the next iteration in the other one. Provided that the processing order is indifferent, this memory is implemented by using two LIFOs as there is no transport delay (compared to a FIFO). The reading/writing direction is controlled by using a signal `switch` which commutes at the end of every iteration.

Note, that despite asynchronous execution, the PDE-based algorithms have one or more synchronization points: the end of the iteration. This is indicated by either: i) emptiness of one of the LIFOs (Narrow Band type algorithms), or ii) end of the raster scan of the image (Global Scope type algorithms). The end of the algorithm is indicated by either: i) emptiness of both LIFOs (for the Narrow Band type algorithms), or ii) the number of necessary iterations (both algorithm types), or iii) the convergence (both algorithm types).

4. RESULTS: CONTINUOUS WATERSHED IMPLEMENTATION

Recall that in terms of PDEs, the watershed transformation can be obtained by computing the weighted distance function to a given set of sources (must be identical to the set of local minima in the image) [12], [13]. In our implementation we have used a parallel algorithm called Massive Marching [5]. The Massive Marching can be used both for narrow band and global scope algorithms without any loss of efficiency. At the same time, it supports massively parallel, semi-parallel or sequential implementation. For the implementation on the proposed architecture, the algorithm can be written as follows:

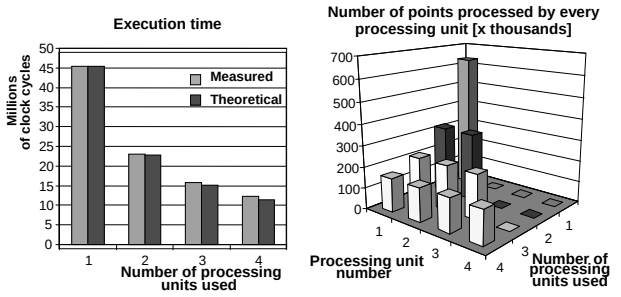
Algo. 2. Continuous watershed by Massive Marching

```
switch=0;
while (LIFO(∅) and LIFO(1) not empty) do
{   // iterations
    for all p_i in LIFO(switch) do (in parallel)
    {   Retrieve_Point p_i from LIFO(switch);
        Retrieve_Neighborhood N(p_i) and p_i;
        Calculate_Value of p_i;
        Update_Value of p_i;// change label and flag
        // activate new points (insertion in A):
        Insert_Other_Points in LIFO(not switch);  }
    switch = not switch;   }
```

We have computed the continuous watershed on one to four parallely operating Processing Units. Figure 5(a) shows the execution time (in terms of total clock cycles versus the number of processing units operating in parallel). The measured number of clocks (including the simultaneous propagation of region labels) is compared to the theoretical execution time in clock cycles obtained as $\text{clk}_N = \text{clk}_1/N$. Although the simple memory access control using semaphores introduces some latency, the measured number of clock cycles is close to the theoretical expected gain.

Figure 5(b) gives the computational load distributed over the processing units. The computational load is expressed as the number of points processed by every PU. The total computational load is uniformly distributed between all PUs.

Table 1 compares the bandwidth of the computation of a weighted distance, with simultaneous propagation of source



(a) (b)

Fig. 5. (a) The execution time of the algorithm in function of the number of parallel Processing Units. (b) The activity load distributed over several Processing Units.

Table 1. The obtained bandwidth vs. other platforms.

Platform/Frequency	Bandwidth ($pixels/s$)
4 RISC cores/120MHz	52.2×10^5
PentiumIII(Linux)/450MHz	29.5×10^5
StrongARM(iPAQ)/206MHz	10.1×10^3

labels, obtained by using Massive Marching implemented on various platforms. The bandwidth is computed as the number of processed points divided by the execution time. Recall that its calculation complexity slightly exceeds $\mathcal{O}(N)$ because some points can be computed several times [5].

5. CONCLUSIONS

This paper proposes an asynchronous multi-core architecture for Level Set formulation of PDE-based algorithms. The implementation choices are demonstrated by the analysis of general algorithms and their timing. The architecture is built around four cores of general-purpose RISC processors and is therefore easily programmable, and re-usable for other purposes.

The measurements and tests of the proposed architecture has been obtained on the implementation of continuous watershed with simultaneous propagation of labels.

It has been shown that the management of the simultaneus memory acceses by simple semaphores is sufficient at least up to four PUs. As results from the study of the execution time and the processing unit activity, the semaphores are efficient and more complicated architecture would be useless. Although the proposed architecture is an embedded system, our benchmarking has shown that it outperforms other embedded systems and its performance is comparable to Pentium based PC.

The future work will focus on two domains: 1) the implementation of other curve evolution algorithms such as deformable models, 2) the optimisation of the processor cores such as the personalization of the instruction set (if the neighbourhood extraction is reduced to one clock cycle, an additional estimated gain of one third of the execution time can be obtained).

6. REFERENCES

[1] A. Sofou and P. Maragos, "PDE-based modeling of image segmentation using volumic," in *IEEE ICIP03*, September 2003.

[2] X. Bresson, P. Vandergheynst, and J.P. Thiran, "A priori information in image segmentation: Energy functional based on shape statistical model and image information," in *IEEE ICIP03, Spain*, September 2003.

[3] F. Guichard and J.M. Morel, "Introduction to partial differential equations in image processing," in *Tutorial Notes, IEEE Int. Conf. Image Proc., Washington*, October 1995.

[4] J. Weickert, B. M. ter Haar Romeny, and M. A. Viergver, "Efficient and reliable schemes for nonlinear diffusion filtering," in *IEEE Transactions on Image Processing*, vol. 7, pp. 398–410. March 1998.

[5] E. Dejnožková and P. Dokládal, "A parallel architecture for curve-evolution PDEs," *Image Analysis and Stereology*, vol. 22, 2003, June.

[6] Y. Tsai, "Rapid and accurate computation of the distance function using grids," Tech. Rep. 17, University of California, USA, 2000.

[7] T. Gijbels, P. Six, L. Van Gool, F. Catthoor, H. De Man, and A. Oosterlinck, "A VLSI architecture for parallel non-linear diffusion with applications in vision," *IEEE, Workshop on VLSI Sig. Proc.*, 1994.

[8] M. Rumpf and R. Strzodka, "Level set segmentation in graphics hardware," *IEEE ICIP01*, 2001.

[9] G. Sapiro, *Geometric Partial Differential Equations and Image Analysis*, Cambridge University Press, 2000.

[10] R. Kimmel, *Curve Evolution on Surfaces*, Ph.D. thesis, Technion Israel Institute of Technology, May 1995.

[11] L. Alvarez, J. Weickert, and Javier Sanchez, "A scale-space approach to nonlocal optical flow calculations," in *Scale-Space Theories in Computer Vision*, 1999, pp. 235–246.

[12] L. Najman and M. Schmitt, "Watershed of a continuous function," *Signal Processing*, vol. 38, pp. 99–112, July 1994.

[13] F. Meyer and P. Maragos, "Multiscale morphological segmentations based on watershed, flooding, and Eikonal PDE," in *Scale-Space Theories in Computer Vision*, pp. 351–362. Springer-Verlag, 1999.