Distributed Computation of Optical Flow

Antonio G. Dopico[1], Miguel V. Correia[2], Jorge A. Santos[3], and Luis M. Nunes[4]

[1] Fac. de Informatica, U. Politecnica de Madrid, Madrid. dopico@fi.upm.es
[2] Inst. de Engenharia Biomédica, U. do Porto, Fac. de Engenharia. mcorreia@fe.up.pt
[3] Inst. de Educacao e Psicologia, U. do Minho, Braga. jas@iep.uminho.pt
[4] Dirección General de Tráfico, Madrid. argos@dgt.es

Abstract. This paper describes a new parallel algorithm to compute the optical flow of a video sequence. A previous sequential algorithm has been distributed over a cluster. It has been implemented in a cluster with 8 nodes connected by means of a Gigabit Ethernet. On this architecture, the algorithm, that computes the optical flow of every image on the sequence, is able of processing 10 images of 720x576 pixels per second.

Keywords: Optical Flow, Distributed Computing

1 Introduction

There is a wide variety of areas of interest and application fields (visual perception studies, scene interpretation, motion detection, filter for in-vehicle inteligent systems etc.) that can benefit from optical flow computing. The concept of optical flow derives form a visual system concept analogue to human retina, in which a 3d world is represented in a 2d surface by means of an optical projection. In the present case we will use a simplified 2d representation consisting in a matrix of n pixels in which only grey values of image are considered. Spatial motion and velocity is then represented as a 2d vector field showing the distribution of velocities of apparent motion of the brightness pattern of a dynamic image.

The optical flow computation of a moving sequence is an intensive demanding application both in memory and computational terms. As the computers performance improves the users expectations raises too: higher resolution video recording systems allow to reduce the negative effects of spatial and temporal motion aliasing. In [1] synthetic images with 1312x2000 pixels at 120 Hz are used. Given the growing need of computer performance the parallelization of the optical flow computation appears to be the only alternative to achieve a massive processing of long video sequences.

This idea of parallelization was proposed some years ago, [2], with four processors, obtained very modest results: processing up to 7-8 images of 64x64 pixels per second, too small resolution to be useful.

More recently [3] proposes the decomposition of the optical flow computation in small tasks: by dividing the image in independent parts the parallelization becomes easier to approach, although with the drawback of the overheads associated with dividing the images and grouping the obtained results. As this has not been yet implemented no results are available.

A possible alternative to parallellization could be to simplify drastically the optical flow algorithm. [4] Presents an alternative based on additions-subtractions that needs much less computational resources but, according to the authors, with the pay-off of incorrect results.

In the present work the parallelization of the optical flow computation is approached with the objective of maximizing performance with no lost of the quality of the results, allowing massive computing of long sequences using standard image resolutions. The gain due to parallelization will be referred to a sequential version of an equivalent algorithm.

2 Optical Flow Computation Sequential Algorithm

Following the survey of Barron et al., [5], the method of Lucas, [6,7], has been chosen to compute optical flow. This method seems to provide the best estimate with less computational effort. Fleet's method, [8], would possibly provide an even better estimate, but the computational cost would be higher due to the use of several Gabor spatio-temporal filters [5].

2.1 Lucas Optical Flow Algorithm

In Lucas's method, optical flow is computed by a gradient based approach.

It follows the common assumption that image brightness remains constant between time frames:

$$I\left(x,y,t\right)=I\left(x+u\delta t,y+v\delta t,t+\delta t\right) \tag{1}$$

which, by also assuming differentiability and using a Taylor series expansion, can be expressed by the motion constraint equation:

$$I_x u\delta t+I_y v\delta t+I_t\delta t=\mathcal{O}\left(u^2\delta t^2,v^2\delta t^2\right) \tag{2}$$

or, in more compact form (considering δt as the temporal unity):

$$\nabla I\cdot\mathbf{v}+I_t=\mathcal{O}\left(\mathbf{v}^2\right) \tag{3}$$

where $\mathcal{O}\left(\mathbf{v}^2\right)$ represents second order and above terms.

In this method, the image sequence is first convolved with a spatio-temporal Gaussian to smooth noise and very high contrasts that could lead to poor estimates of image derivatives. Then, according to Barron et al. implementation, the spatio-temporal derivatives I_x, I_y and I_t are computed with a four-point central difference.

Finally, the two components of velocity $\mathbf{v}=(u,v)$ are obtained by a weighted least-squares fit of local first-order constraints, assuming a constant model for $\mathbf{v}$ in each small spatial neighborhood $\mathcal{N}$, by minimizing:

$$\sum_{x\in\mathcal{N}}\mathbf{W}^2\left(\mathbf{x}\right)\left[\nabla I\left(\mathbf{x},t\right)\cdot\mathbf{v}+I_t\left(\mathbf{x},t\right)\right]^2 \tag{4}$$

where $\mathbf{W}(\mathbf{x})$ denotes a window function that weights more heavily at the centre. The solution results from:

$$\mathbf{v} = (\mathbf{A}^T\mathbf{W}^2\mathbf{A})^{-1}\mathbf{A}^T\mathbf{W}^2\mathbf{b} \tag{5}$$

where, for n points $\mathbf{x}_i \in \mathcal{N}$ at a single time t,

- $\mathbf{A} = [\nabla I(\mathbf{x}_1), ..., \nabla I(\mathbf{x}_n)]^T$,
- $\mathbf{W} = diag\,[\mathbf{W}(\mathbf{x}_1), ..., \mathbf{W}(\mathbf{x}_n)]$ and
- $\mathbf{b} = -(I_t(\mathbf{x}_1), ..., I_t(\mathbf{x}_n))^T$.

The product $\mathbf{A}^T\mathbf{W}^2\mathbf{A}$ is a 2×2 matrix given by:

$$\mathbf{A}^T\mathbf{W}^2\mathbf{A} = \begin{bmatrix} \sum \mathbf{W}^2(\mathbf{x}) I_x^2(\mathbf{x}) & \sum \mathbf{W}^2(\mathbf{x}) I_x(\mathbf{x}) I_y(\mathbf{x}) \\ \sum \mathbf{W}^2(\mathbf{x}) I_y(\mathbf{x}) I_x(\mathbf{x}) & \sum \mathbf{W}^2(\mathbf{x}) I_y^2(\mathbf{x}) \end{bmatrix} \tag{6}$$

where all sums are taken over points in the neighborhood $\mathcal{N}$.

Simoncelli [9,10] present a Bayesian perspective of equation 4. They model the gradient constraint using Gaussian distributions. This modification allows to identify unreliable estimates using the eigenvalues of $A^T W^2 A$.

2.2 Implementation

Now the sequential implementation of the Lucas-Kanade algorithm is explained [11,12]:

- The implementation first smoothes the image sequence with a spatiotemporal Gaussian filter to attenuate temporal and spatial noise as do Barron et al. [5]:
  - Temporal smoothing Gaussian filter with $\sigma = 3.2$, requiring $6\sigma + 1$ (21) frames, the current frame, 3σ (10) past frames and 3σ (10) future frames.
  - Spatial smoothing Gaussian filter with $\sigma = 3.2$, requiring $6\sigma + 1$ (21) pixels, the central pixel and 3σ (10) pixels for each side relative to this central pixel. This symmetric Gaussian filter in one dimension is applied twice, first in the X dimension and then in the Y dimension.
- After the smoothing, spatiotemporal derivatives (I_t, I_x, I_y) are computed with 4-point central differences with mask coefficients:

$$\frac{1}{12}(-1, 8, 0, -8, 1) \tag{7}$$

- Finally, the velocity is computed from the spatiotemporal derivate:
  - A spatial neighborhood of 5x5 pixels is used for the velocity calculations.
  - A weight matrix identical to Barron [5], i.e., with 1-D weights of $(0.0625, 0.25, 0.375, 0.25, 0.0625)$ is also used for the velocity calculations.
  - The noise parameters used are $\sigma_1 = 0.08$, $\sigma_2 = 1.0$, $\sigma_p = 2.0$ ([9]).
  - Velocity estimates where the highest eigenvalue of $A^T W^2 A$ is less than 0.05 is considered unreliable and removed from the results ([5]).

2.3 Sequential Algorithm Results

Figure 1.a shows an image of a interlaced video sequence with 720x576 pixels, that have been processed with the described algorithm. The optical flow obtained is shown in figure 1.b. The car on the left is going faster than the car on the center and the car on the right is going slower than the car on the center.

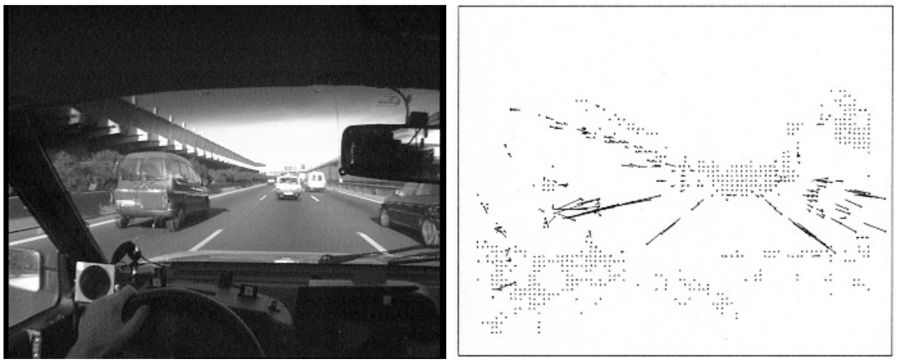

Fig. 1. Video Sequence: Frames 19 and 29.

3 Parallelization of the Optical Flow Computing

The parallelization of the sequential algorithm is explained in this section.

3.1 Parallel Algorithm

The execution time of the different tasks of the sequential algorithm have been measured to obtain an estimation of its weights. The measures have been obtained using a workstation with an Intel Xeon 2.4 GHz and 1GB of main memory, though the important data are not the absolute times but the relationship among the different tasks.

- The temporal smooth, in T, is slower than the others because it works with a high number of images. Moreover it has to read them from the disk (12 ms).
- The spatial smooth in X employs 8 ms.
- The spatial smooth in Y employs 7 ms. Probably the difference is because now the image is in the cache memory.
- Computation of the partial derivatives, (It,Ix,Iy), 10 ms.
- Computation of the velocity of each pixel and writing the results to disk, 130 ms. This is more than triple the time spent by the rest of the tasks.

These times are spent with each image in the video sequence.

Unlike [3] the images have not been divided to avoid the introduction of unnecessary overheads, because in that case they had to be divided, then processed and finally group the results. Moreover, the possible boundary effects should be taken into account. Anyway, this option could be useful in some cases.

To structure the parallelization, the existing tasks have been taken into account. The first four tasks are connected as a pipeline because they need the data of several images to work properly. The last one only needs a single image and it is actually independent. The fourth task will send derivatives from complete images to different copies of task five in a rotative way.

Although a 8 nodes cluster has been used for the implementation, the followed schema is flexible enough to be adapted to different situations:

– Four nodes. The first one executes all the tasks except computing the velocity of the pixels (37 ms). The rest of the nodes compute the velocities, when they finish with an image, they start with the next one (130/3 = 43 ms per node). One image would be processed every 43 ms (maximum of 37 and 43).
– Eight nodes. The first node computes the temporal smooth and the spatial smooth for the X co-ordinate (12+8=20ms). The second one computes the spatial smooth for the Y co-ordinate and the partial derivatives (7+10=17ms). The rest of the nodes compute the velocities (130/6=21ms). An image is processed every 21 ms (maximum of 20, 17 and 21).
– Sixteen nodes. The first four nodes are dedicated to the first four tasks (12, 8, 7 and 10 ms respectively). The rest of the nodes compute the velocities (130/12=11ms). An image would be processed every 12ms (maximum of 12, 8,7,10,11).

In the three cases, the communication time has to be added. This time would depend on the net (Gigabit, Myrinet, etc.) but in every case it has to be taken into account and it will employ several milliseconds.

With this scheme, even if a cluster is not used it would not be a problem. For example, a shared memory tetraprocessor could be used and the tasks could be distributed in the same way than with a four nodes cluster.

With more than 16 nodes, there are not enough tasks to distribute. To obtain a higher degree of parallelism the images would be divided as [3] proposes. Each subimage would be independent of the rest if some boundary pixels are added. That is, as the spatial smooth uses 25 pixels, the central one and 12 on each side, each subimage would need 12 pixels more per boundary. So, to divide images of 1280x1024 pixels in 4 subimages (2x2) they should be divided in regions of 652x524 pixels, with overlapped boundaries. In this way each image would be totally independent.

3.2 Cluster Architecture

A cluster with 8 biprocessor nodes (2.4 GHz, 1GB RAM) running Linux (Debian with kernel 2.4.21) and openMosix has been used. The nodes are connected using a Gigabit Ethernet switch. This distributed memory architecture was chosen because it is not expensive, it is easy to configure and it is broadly extended.

3.3 Implementation

The tasks of the previously described algorithm have been assigned to the different nodes of the cluster. For communications, the message passing standard MPI, in short, the open source implementation LAM/MPI version 6.5.8. of the University of Indiana, has been used.

For the mentioned communications, non blocking messages have been used, in such a way that the computation and the communications are overlapped. Moreover, the use of persistent messages avoids the continuous creation and destruction of the data structures used by the messages. This has been possible because the communication scheme is always the same. The information that travels between two given nodes has always the same structure and the same size so, the message backbone can be reused.

About the non blocking messages, a node, while processing the image `i`, has already started a non blocking sending to transfer the results of processing the previous image `i-1` and has also started a non blocking reception to simultaneously gather the next image `i+1`. This allows simultaneously send, receive and compute in each node.

About the task distribution among the nodes, the scheme has been the following:

- **Node 1**. Executes the following tasks:
  - Reads the images of the video sequence from the disk.
  - Executes the temporal smooth. To do that, the current image and the twelve previous ones are used
  - Executes the spatial smooth for the `x` co-ordinate.
  - Sends to the node 2 the image smoothed in `t` and `x`.
- **Node 2**. Executes the following tasks:
  - Receives the image from node 1.
  - Executes the spatial smooth for the `y` co-ordinate.
  - Computes the partial derivative in `t` of the image. To do that five images are used, the current one, the two previous and the two next. So, if the image `i` is received, the derivative in `t` of the image `i-2` is computed.
  - Computes the partial derivatives in `x` and `y` of the image.
  - Sends the computed derivatives `It`, `Ix` and `Iy` to the next nodes (from 3 to 8) in a cyclic mode. When the node 8 is reached, it starts again in the node 3.
- **Rest of the nodes**. They execute the following tasks:
  - Receive the partial derivatives in `t`, `x` and `y` of the image, `It`, `Ix` and `Iy`.
  - Using the derivatives, computes the velocity of each pixel as (vx, vy).
  - Write in the disk the computed velocities.

Figure 2 shows the distribution of the tasks among the nodes.

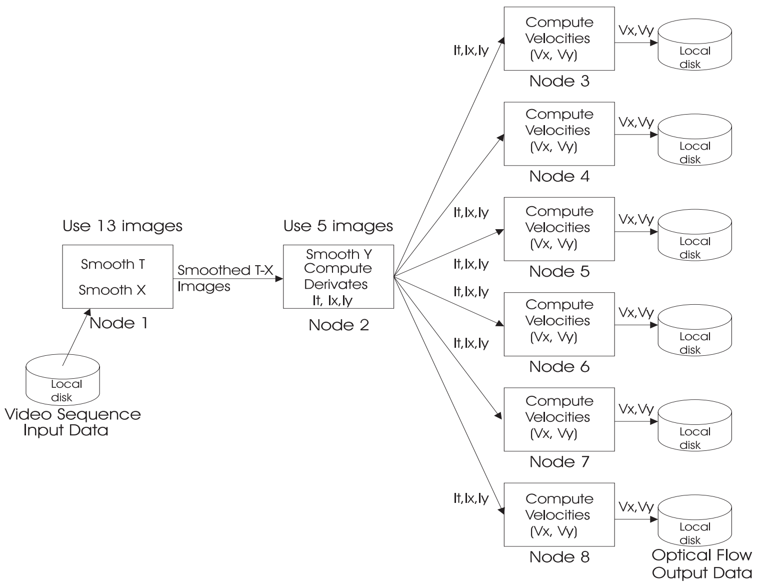

Fig. 2. Tasks Distribution.

3.4 Results

With this parallelization scheme and using the above described cluster, the computation of the optical flow is achieved at 30 images per second with images of 502x288 pixels. For images of 720x576 pixels the obtained speed is 10 images per second. Note that the optical flow, in both cases, is computed for every image in the video sequence without skipping any one. This performance means a speedup of 6 over the sequential version employing 8 nodes.

4 Conclusions and Future Works

This paper presents a new distributed algorithm for computing the optical flow of a video sequence. This algorithm is based on the balanced distribution of its tasks among the nodes of a cluster of computers. The distribution done is flexible and can be adapted to several environments, with shared memory as well as with distributed memory. Moreover, it is easily adaptable to a wide range of nodes number: 4, 8, 16, 32 or more.

The algorithm has been implemented on a cluster with 8 nodes and a gigabit Ethernet, where 30 images per second can be processed with images of 502x288 pixels, or 10 images per second if the images are of 720x576 pixels. With respect to the sequential version the resulting speedup is 6 times faster.

Taking into account the modest performance obtained in [2] with four processors (6-7 images per second with images of 64x64 pixels), or the inconvenients of the simplified algorithms [4] the results obtained with the algorithm proposed here are very hopeful. The interesting parallelization [3] cannot be compared because it is not yet implemented.

The obtained performance brings important advantages. Working with longer sequences, larger images (1280x1024 pixels or even larger) and higher frequencies is now feasible. Increased temporal resolution is particularly beneficial in complex scenarios with high speed motion. In this line, the particular motion aliasing pattern of the current interlaced cameras can be reduced by an additional algorithm that duplicates the frequency and may be helpful prior the optic flow computation: the video sequence can be rebuilt by combining each half frame both with the prior and with the next half-frame, (hf1+hf2); (hf2+hf3); (hf3+hf4); (hf4+hf5), and so on. The result is an upgraded sequence with less motion aliasing and double temporal frequency.

Regarding real time applications, by connecting the video signal directly to one of the nodes of the cluster and digitizing the video sequence on the fly, the current implementation of the algorithm allows on line optical flow calculation of images of 502x288 pixels at 25 to 30 Hz.

References

1. Lim, S., Gamal, A.: Optical flow estimation using high frame rate sequences. In: Proceedings of the International Conference on Image Processing (ICIP). Volume 2. (2001) 925–928
2. Valentinotti, F., Di Caro, G., Crespi, B.: Real-time parallel computation of disparity and optical flow using phase difference. Machine Vision and Applications **9** (1996) 87–96
3. Kohlberger, T., Schnörr, C., Bruhn, A., Weickert, J.: Domain decomposition for parallel variational optical flow computation. In: Proceedings of the 25th German Conference on Pattern Recognition, Springer LNCS. Volume 2781. (2003) 196–202
4. Zelek, J.: Bayesian real-time optical flow. In: Proceedings of the 15th International Conference on Vision Interface. (2002) 266–273
5. Barron, J., Fleet, D., Beauchemin: Performance of optical flow techniques. International Journal of Computer Vision **12** (1994) 43–77
6. Lucas, B.: Generalized Image Matching by Method of Differences. PhD thesis, Department of Computer Science, Carnegie-Mellon University (1984)
7. Lucas, B., T., K.: An iterative image registration technique with an application to stereo vision. In: Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI). (1981) 674–679
8. Fleet, D., Langley, K.: Recursive filters for optical flow. IEEE Transactions on Pattern Analysis and Machine Intelligence **17** (1995) 61–67
9. Simoncelli, E., Adelson, E., Heeger, D.: Probability distributions of optical flow. In: IEEE Conference on Computer Vision and Pattern Recognition. (1991) 310–315
10. Simoncelli, E.: Distributed Representation and Analysis of Visual Motion. PhD thesis, Massachusetts Institute of Technology (1993)
11. Correia, M., Campilho, A., Santos, J., Nunes, L.: Optical flow techniques applied to the calibration of visual perception experiments. In: Proceedings of the Int. Conference on Pattern Recognition, 13 ICPR. Volume 1. (1996) 498–502
12. Correia, M., Campilho, A.: Real-time implementation of an optical flow algorithm. In: Proceedings of the Int. Conference on Pattern Recognition, 16th ICPR, Volume IV. (2002) 247–250