Fast-Coding Robust Motion Estimation Model in a GPU

Carlos García[a]

Guillermo Botella[a]

Francisco de Sande[b]

Manuel Prieto-Matías[a]

[a]Complutense University of Madrid, Spain; [b]La Laguna University, Spain;

ABSTRACT

Nowadays vision systems are used with countless purposes. Moreover, the motion estimation is a discipline that allow to extract relevant information as pattern segmentation, 3D structure or tracking objects. However, the real-time requirements in most applications has limited its consolidation, considering the adoption of high performance systems to meet response times. With the emergence of so-called highly parallel devices known as accelerators this gap has narrowed. Two extreme endpoints in the spectrum of most common accelerators are Field Programmable Gate Array (FPGA) and Graphics Processing Systems (GPU), which usually offer higher performance rates than general propose processors. Moreover, the use of GPUs as accelerators involves the efficient exploitation of any parallelism in the target application. This task is not easy because performance rates are affected by many aspects that programmers should overcome. In this paper, we evaluate OpenACC standard, a programming model with directives which favors porting any code to a GPU in the context of motion estimation application. The results confirm that this programming paradigm is suitable for this image processing applications achieving a very satisfactory acceleration in convolution based problems as in the well-known Lucas & Kanade method.

Keywords: Motion Estimation, GPU, OpenACC

1. INTRODUCTION

In the modern era, mankind has tried to endow machines, through the computerization process, with a greater similarity to humans so that machines can help in a better way and perform a larger number of tasks. To achieve this goal, machines are being equipped with similar senses to humans, facilitating their interaction with the world around us. One of the senses which has undergone the biggest advances over the last few years is the sense of vision,[1–4] in an attempt to give machines the ability that humans have for reading the environment and extracting a large amount of information from it. This interest of the human race in producing machines with senses such as sight is explained by the ease and the speed with which a machine can analyze information from the environment.

As part of the sense of vision, we find the motion estimation paradigm becoming of capital importance for a huge number of applications, such as sport tracking, surveillance, quality control, navigation, cybernetics, medicine and so on.

An illustrative example in the biomedical field is that of breast MRI, where the most significant information is acquired from differences in images taken pre-contrast and post-contrast.[5,6] Since we have the handicap that a typical breast MRI reading takes several minutes, it is highly likely that there will be motion between the two readings used to compute a difference image, due to the patient moving as well as to sagging breast tissue. Therefore, many motion artifacts are caused by involuntary patient motion or muscle relaxation during the image acquisition, affecting any image analysis performed on such perturbed images. A reduction of such artifacts is thus fundamental for a valid morphological or dynamic image analysis.

Further author information: (Send correspondence to C. García): E-mail: garsanca@ucm.es

Real-Time Image and Video Processing 2015, edited by Nasser Kehtarnavaz, Matthias F. Carlsohn,
Proc. of SPIE-IS&T Electronic Imaging, SPIE Vol. 9400, 94000J · © 2015 SPIE-IS&T
CCC code: 0277-786X/15/$18 · doi: 10.1117/12.2077091

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 07/16/2015 Terms of Use: http://spiedl.org/terms

Optical flow is a term that has been widely linked to motion estimation, despite not being exactly the same concept. The brightness pattern in the image moves as the object that appears in it moves. Optical flow is the apparent motion of the brightness pattern of entities such as surfaces, edges, and objects in a visual scene,[7] caused by the relative motion between an observer (an eye or a camera) and the scene . In the ideal case, optical flow would correspond with the estimated motion field, despite the fact that apparent motion can be caused by lighting changes without any actual motion. Optical flow involves a huge and complex signal processing operation which represents a handicap when a real-time constraint is needed.

Due to the drawback of having to provide real-time results, many approaches to accelerating optical flow algorithms by means of devices such as GPUs have been considered. These devices are based on multi-core systems with a complex memory hierarchy. These platforms are designed to take advantage of the inherent parallelism of data rendering 3D scenes. However, currently these platforms are used as parallel coprocessors, executing a high number of threads simultaneously

The programming paradigm represents the GPU as a coprocessor, which can execute parallel kernels and also offers extensions for the C language. This paradigm maps the data from the GPU, transfers the data between the GPU and CPU, and runs the kernels. Usually a kernel runs several code lines over many parallel threads. This kind of system uses the concept of SIMT (Single Instruction Multiple Threads), so a single instruction is executed from many threads with different input data. The tasks are organized through the so -called blocks, which make it possible to run a huge number of cooperating threads, thanks to a low-latency local memory and the use of synchronization tokens. Some blocks can only be coordinated through the global memory with high latency. Processors are grouped into multiprocessors, which are assigned to a particular task.[8]

Regarding a specific paradigm to program the GPU we have used the novel paradigm OpenACC,[9] it being an accelerator programming standard that enables C -based programmers to directly take advantage of the power of heterogeneous CPU/accelerator systems. This paradigm thus allows programmers to use compiler directives to identify which areas of code to accelerate, without requiring modification to the underlying code itself. By identifying parallel code segments, OpenACC directives allow the compiler to do the detailed work of mapping the computation onto the accelerator.

This paper is organized as follows, after the present section. Section 2 addresses the Optical Flow algorithm to be accelerated. Section 3 presents the specific directive -based paradigm that will be used to program the accelerator. Section 4 explains the implementation section by means of choosing candidate functions. Section [sec:results] shows the results obtained and finally Section 6 contains the conclusion and future lines of research.

2. LUCAS & KANADE ALGORITHM

The estimation of the velocity field using optical flow is an ill-posed problem, since there are an infinite number of velocity fields that can cause the observed changes in the luminance distribution. Additionally, there are an infinite number of 3-dimensional motions in the real world that could yield a particular velocity field. External knowledge about the behavior of objects in the real world, such as rigid body motion constraints, are required in order to make use of optical flow. Despite the problems, the optical flow information is a rich array of vectors that has both local and global properties.[10] The optical flow field can thus be subjected to many higher-level interpretations.[11]

The Lucas & Kanade method[12] is a well-known algorithm, and we have applied the original description of the model,[13] while adding several variations to improve the viability of the hardware implementation. We present a simplified scheme of the algorithm, below. The Lucas & Kanade model computes optical flow using a gradient technique that makes use of space-temporal derivative filters. The model comes from the basic intensity conservation over time (equation 1), where x, y, and t are the coordinates of the sequence. Developing the expression 1 we reach expression 2:

$$\frac{dI\left(x(t), y(t), t\right)}{dt} = 0 \tag{1}$$

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 07/16/2015 Terms of Use: http://spiedl.org/terms

$$
\begin{aligned}
&\frac{dI\left(x(t),y(t),t\right)}{dt} = \\
&\frac{\delta I\left(x(t),y(t),t\right)}{\delta x}\frac{dx}{dt} + \frac{\delta I\left(x(t),y(t),t\right)}{\delta y}\frac{dy}{dt} + \frac{\delta I\left(x(t),y(t),t\right)}{\delta t} = \\
&\frac{\delta I\left(x(t),y(t),t\right)}{\delta x}u + \frac{\delta I\left(x(t),y(t),t\right)}{\delta y}v + \frac{\delta I\left(x(t),y(t),t\right)}{\delta t} = 0
\end{aligned}
\tag{2}
$$

The Lucas and Kanade model assumes that the motion vector really does not change in the studied vicinity V. Considering the error to minimize the motion constraint expression (equation 3):

$$
E(u,v) = \sum_{pixel \in V} \left(\frac{\delta I\left(x(t),y(t),t\right)}{\delta x}u + \frac{\delta I\left(x(t),y(t),t\right)}{\delta y}v + \frac{\delta I\left(x(t),y(t),t\right)}{\delta t} \right)^2 \tag{3}
$$

Solving $\frac{\delta E}{\delta u(t)} = 0; \frac{\delta E}{\delta v(t)} = 0$; and grouping them all together, we find an algebraic system expressed by equation 4, which means the least mean square estimation of the optical flow at the centered pixel of the vicinity V. The symbol $\wedge$ denotes the estimator of the corresponding magnitude. The resulting optical flow estimated is dense:

$$
\begin{aligned}
&\begin{bmatrix} \hat{u} \\ \hat{v} \end{bmatrix} \cdot A = B \\
&A = \begin{bmatrix} \sum_{pixel \in V} I_x^2(pixel) & \sum_{pixel \in V} I_x I_y(pixel) \\ \sum_{pixel \in V} I_x I_y(pixel) & \sum_{pixel \in V} I_y^2(pixel) \end{bmatrix} \\
&B = \begin{bmatrix} \sum_{pixel \in V} I_x I_t(pixel) \\ \sum_{pixel \in V} I_y I_t(pixel) \end{bmatrix}
\end{aligned}
\tag{4}
$$

where a sub-index in the equation means the derivatives computed by separable filtering (Gaussian derivatives or Gabor function), so I_x, I_y and I_t correspond to $\frac{\delta I(x(t),y(t),t)}{\delta x}$, $\frac{\delta I(x(t),y(t),t)}{\delta y}$ and $\frac{\delta I(x(t),y(t),t)}{\delta t}$ respectively.

3. OPENACC PROGRAMMING

Currently, OpenACC[9] is one of the most prominent Directive-Based Programming Models, as it is being established as a standard for parallel computing for different accelerators. It was developed by the group of companies PGI, CAPS, NVDIA and CRAY. Its objective is to simplify the programming of heterogeneous systems (CPU/GPU), to facilitate code migration and to be an intermediate step between automatic parallelization and manual parallelization.

OpenACC is a directive programming model. The programmer inserts a set of directives to guide the compiler concerning which parallel regions could be executed in the accelerator.

Regarding the OpenACC execution model, it is based on a *host-device* model as in OpenCL or CUDA. The work-flow is managed by the *host*, and the *device* is responsible for executing parallel regions. The *host* is in charge of memory allocation on the device, sending data, downloading code onto the accelerator and sending the required arguments in the parallel region. It is also responsible for maintaining a queue with the code that will be executed on the accelerator and for bringing back the data and results to the *host*.

The programmer is responsible for exploiting different parallelisms (fine-grained, coarse-grained, SIMD or vector operations). OpenACC divides the workload into *gangs*, *workers* and *vector*, which may vary depending on the compiler and target platform. These concepts are linked with the CUDA/OpenCL architectural model.

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 07/16/2015 Terms of Use: http://spiedl.org/terms

One or more *gangs* are launched on the multiprocessor *device*; these gangs being composed of several workers, which are maps to a single processing unit called a core in CUDA.

OpenACC includes directives to define parallel regions: *kernel* or *parallel* construction. While *kernel* construction resembles the description of kernel in OpenCL or CUDA, *parallel* indicates that source-code does not present data and control dependencies so could be mapped in a parallel architecture.

In order to illustrate the ease of coding with OpenACC, the well-known *convolution* operation is used. The *convolution* CUDA version is shown in Figure 1. Previously device memory initialization (*cudaMemcpy* and *cudaMalloc* operations) is needed; later, the execution on the device is performed by "kernel" invocation; and finally, results are returned to host (last *cudaMemcpy* call).

```
__global__ void conv1D(float *conv, float *im, float *filter,
   int width, int height, int filter_size)
{
   // Block index
   int bx = blockIdx.x;
   int by = blockIdx.y;

   // Thread index
   int tx = threadIdx.x;
   int ty = threadIdx.y;

   // Index processed by the block
   int i = BLOCK_SIZE*by + ty;
   int j = BLOCK_SIZE*bx + tx;

   int bi, bj;
   int filter_center = (filter_size -1)/2;

   conv[i*width+j] = 0.0;
   for (bj=-filter_center; bj<=filter_center; bj++)
      if ( (j+bj>=0) & (j+bj<nx) )
         conv[i*width+j] += im[i*width+j+bj]*filter[bj+filter_center];

}

int main(int argc, char** argv)
{
   ....
  cudaMalloc((void**)&d_im, width*height*sizeof(float));
  cudaMalloc((void**)&d_conv, width*height*sizeof(float));
  cudaMalloc((void**)&d_filter, filter_size*sizeof(float));
  cudaMemcpy(d_im, im, width*height*sizeof(float), cudaMemcpyHostToDevice);
  cudaMemcpy(d_filter, filter, filter_size*sizeof(float), cudaMemcpyHostToDevice);

  /* CUDA Kernel */
  conv1D<<< 128, 256>>>(d_conv, d_im, d_filter, width, height, filter_size);

  cudaMemcpy(d_conv, conv, width*height*sizeof(float), cudaMemcpyHostToDevice);
  ...
}
```

Figure 1. Source code for *convolution* operation wrote in NVIDIA-CUDA.

As can be seen in Figure 1, performing a kernel involves rewriting most of the source code with implicit device memory management. Nevertheless, the use of the OpenACC paradigm simplifies the whole encoding process.

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 07/16/2015 Terms of Use: http://spiedl.org/terms

As illustrated in Figure 2, it is only necessary to insert *#pragma acc kernels loop* where the *im* and *filter* array are copied to device and *conv* is copied back to host.

```
int main(int argc, char** argv)
{
   ....

#pragma acc data copyin(im[0:width*height],filter[0:filter_size])\
   copyout(conv[0:width*height])
{
   #pragma acc kernels loop independent
   for (i=0; i<height; i++){
      #pragma acc loop independent
      for (j=0; j<width; j++){
         conv[i*width+j] = 0.0;
         #pragma acc loop seq
         for (bj=-filter_center; bj<=filter_center; bj++)
            if ( (j+bj>=0) & (j+bj<width) )
               conv[i*width+j] += im[i*width+j+bj]*filter[bj+filter_center];
      }
   }
}

   ....
}
```

Figure 2. *Convolution* operation coded using OpenACC paradigm.

4. IMPLEMENTATION

This section addresses the Lucas & Kanade algorithm by means of OpenACC. Candidate functions are selected according to the degree of data parallelism available. These are necessary to adapt the source code in order to exploit accelerator features.

The first transformation performed consists in a reduction of data transfers between *host* and GPU, and a single data region is coded. Sequence images, and filters are uploaded by means of the *copyin* clause. Finally, velocity information is downloaded at the end of the data region. Algorithm 1 shows the main OpenACC codification.

The pseudocode in Algorithm 1 shows the data region that contains the functions to compute the Lucas&Kanade method. As can be seen, data transfers are performed by *copyin*/*copyout* clauses. Furthermore, temporal memory allocations on the GPU are needed in derivatives and summations (*create* clause).

Algorithm 1 also shows the *Horizontal Spacial Derivative* pseudocode. The convolution performs the calculation of derivatives on the x-axis by means of fissioned loops. Most of the code is written using nested loops and conditional branches (in left and right edges filtering). Furthermore, OpenACC kernel construction is defined (*#pragma acc kernel*) with *present* to request that data which is already allocated on the GPU. Loop iterations are also marked as independent by means of the *#pragma acc loop independent* directive. As the inner loop workload does not have a sufficient degree of parallelism to be exploited successfully on a GPU, it is advisable to perform it in a single CUDA-thread way (*#pragma acc loop seq* directive). Although a more common choice would be to fission the loops of the convolutions to avoid conditional branches, OpenACC does not create efficient kernels (in terms of performance). This is due to the fact that loops that perform edges contain very little computational workload in comparison with the main loop, which performs the center frame. Consequently, it is not worth generating kernels without enough parallel data workload.

Finally, we would like to mention two more transformations that provide successful speedups:

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 07/16/2015 Terms of Use: http://spiedl.org/terms

Algorithm 1 $[U, V] = lk(movie, filters)$

```
filter_X = get_hor_filter(filters)
filter_Y = get_vert_filter(filters)
                                                  ▷ ms,fs,fils are movie, frame and filter size
#pragma acc data copyout(U[0:ms], U[0:ms])
    copyin(movie[0:ms], filterX[0:fils], filterY[0:fils])
    create(Ix[0:fs], Iy[0:fs], It[0:fs],
      I_x^2[0:fs], I_y^2[0:fs], I_xt[0:fs], I_yt[0:fs],
      sum_Ix^2[0:fs], sum_Iy^2[0:fs],
      sum_IxIt[0:fs], sum_IyIt[0:fs])
{
for z=0 to nFrames-1 do
    f0 = movie[z * fs]
    f1 = movie[z * fs]
    I_x = spacial_derivative(f0, filterX)
    I_y = spacial_derivative(f0, filterY)
    I_t = temporal_derivative(f0, f1)
    sum_Ix^2 = calculate_sum(I_x, I_x)
    sum_Iy^2 = calculate_sum(I_y, I_y)
    sum_Ixt = calculate_sum(I_x, I_t)
    sum_Iyt = calculate_sum(I_y, I_t)
    [U_z, V_z] = solve_system(sumI_x^2, sumI_y^2, sum_IxIt, sum_IxIt)
end for
}
                                                  ▷ Horizontal Spacial Derivative
function I_x=SPACIAL_DERIVATIVE(frame, filterX)
    #pragma acc kernel present(frame[0:fs], filterX[0:fils],
        I_x[0:fs])
    {
    #pragma acc loop independent
    for i=0 to vs do                              ▷ vs is vertical size
        #pragma acc loop independent
        for j=0 to hs do                          ▷ hs is horizontal size
            tmp = 0
            if j < fs/2 then                      ▷ left border
                #pragma acc loop seq
                for k=0 to fs do
                    tmp+ = frame[i][j + (k − fs/2)] * filterX[k]
                end for
            else if j ≥ hs-fs/2 then              ▷ right border
                #pragma acc loop seq
                for k=0 to fs/2-hs-j do
                    tmp+ = frame[i][j + (k − fs/2)] * filterX[k]
                end for
            else                                  ▷ center
                #pragma acc loop seq
                for k=0 to fs do
                    tmp+ = frame[i][j + (k − fs/2)] * filterX[k]
                end for
            end if
            I_x[i][j] = tmp
        end for
    end for
    }
end function
```


Downloaded From: http://proceedings.spiedigitallibrary.org/ on 07/16/2015 Terms of Use: http://spiedl.org/terms

- The first one consists in making an inline expansion within the kernels (inserting source code instead of a function call), since the compiler does not allow function calls inside these constructions.
- The second one avoids the use of matrix or vector structures because the compiler does not generate a consistent kernel. Data are expressed as scalar

5. RESULTS

5.1 Work Environment

To carry out the tests on the implementation developed in OpenMP *, we have used a system with two Intel Xeon E5530 processors with 4 cores each , at 2.40GHz, with 8MB of cache and Hyperthreading technology; performing such tests in configurations of 2, 4, 8 cores and 8 cores + SMT, and making use of the gcc compiler in its 4.7.2 version . Regarding the implementation of OpenACC, the accelerator used has been a GPU Tesla K20c from NVIDIA (Kepler GK110 architecture). It includes 2496 CUDA cores with 5GB GDDR5 memory and a floating point performance of 3.52Tflops. The OpenACC compiler used was PGI-pgcc, version 14.6.

5.2 Convolution Performance

In this subsection, it's evaluated GPU behavior with the well-known convolution operator using the OpenACC standard. Table 1 show the speedups obtained in a modern NVIDIA GPU using OpenACC standard varying image and filtering size. As a baseline code, we have considered the 2D separable convolution developed in C language. GPU version was performed adding OpenACC's *pragmas* which means inserting only 3% of new source code lines. As was expected, successful accelerations are reached in most configuration with a range of $1\times - 153\times$ for PGI compiler.

Table 1. Speedups achieved in Tesla K20c device using both OpenACC compilers.

Image size	128^2			256^2			512^2			1024^2		
Filter size	7	15	31	7	15	31	7	15	31	7	15	31
CAPS	0.2	0.3	5.7	1.4	2.4	22.9	1.6	9.6	71.7	9.8	25.5	134.1
PGI	1.0	2.5	24.1	3.6	8.0	70.5	9.1	16.5	113.9	18.2	29.7	153.7

We would like to mention that most of motion estimation algorithms are based on temporal and spatial derivatives implemented by convolve filtering.

5.3 Lukas&Kanade accuracy

The first experiment consists in a study of the accuracy of the Lucas & Kanade algorithm. The metric used is the well-known Barron's angle.[14] The synthetic stimuli used are Diverging Tree and Translating Tree.[15] The calculated error is obtained as the average error per pixel between the estimated optical flow and the ground truth.

Error results are performed using a different parameter configuration: varying spatial filter sizes (3, 5, 7 and 9) and block neighbourhood sizes (5, 9, 15, 19, 26 and 30). We have also considered the accuracy for different density ranges (with/without the edge region). As summary, tables 2 and 3 show the best configuration obtained with Diverging and Translating Tree stimuli. The whole frame has 100% density, and a density of 93% for an image of 150×150 with a filter of size 5 corresponds to the following calculation: $(150-5)^2/150^2 = 0.93$. Therefore, in this study we test the trade-off solution between accuracy and density. 'error without correction' notation refers to a density of 100%.

*OpenMP: www.openmp.org

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 07/16/2015 Terms of Use: http://spiedl.org/terms

Table 2. Metric Barron error in DivergingTree for different corrections with their densities.

Type	Block	Filter	Angular Error	Density
Without correction	15	5	14.46^o	100%
Filter correction	15	9	11.67^o	89%

Table 3. Metric Barron error in TranslatingTree for different corrections with their densities.

Type	Block	Filter	Angular Error	Density
Without correction	30	5	26.21^o	100%
Filter correction	30	9	15.11^o	93%

5.4 Lukas&Kanade performance

In the performance study, the most accurate configuration has been chosen (size filter=5 and blocks neighborhood=15). Figure 3 shows the performance measured in frames per second (fps) with varying frame resolutions (from 150^2 to 2400^2) on both multicore CPU and GPU.

As can be observed, the non-parallel version (blue bar denoted as 'Linear') only achieves real-time ratios (about 20 or 25fps) for low resolution rates (150^2). As expected, the exploitation of task-level parallelism on a multicore processor using OpenMP reported better performance. The use of a multicore system with 8 cores+SMT (SMT means Intel's HyperThreading enabled) provides real time response for 600^2 resolutions. However, using the novel OpenACC paradigm, we obtain real time with upto 1200^2 resolutions, getting closer to real time with 14 fps for FullHD resolution (1920×1080). These encouraging results suggest that the use of a single GPU in the context of motion estimation can offer improvements of between $3 - 4\times$ with respect to the best multicore version without significant variations in the programming task.

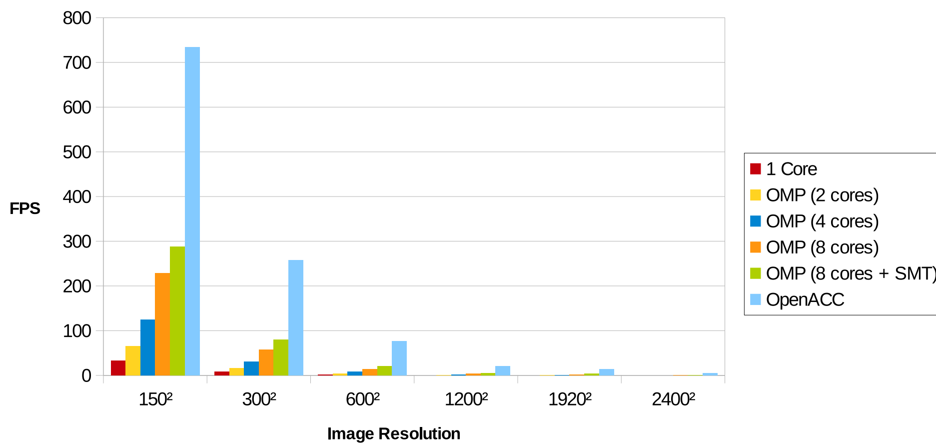

Figure 3. Performance (fps) with different resolutions for DivergingTree using filter=5 and block=15 configuration.

The speedups achieved (see Figure 4) with OpenMP in multicore are in a range between $2 - 10\times$. However, there is a successful scalability of more than 70% when physical cores are available, while that is reduced to 50% with 8 cores+SMT. Nevertheless, the GPU version with OpenACC achieves better speedups, in a range of $22\times$ to $40\times$ with respect to its serial CPU counterpart. We can notice that when increasing the image resolution, OpenMP does not provide a great improvement because it is reaching the maximum parallelism that can be supplied. However, the GPU version with OpenACC continues increasing the speedup in huge images, as GPUs

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 07/16/2015 Terms of Use: http://spiedl.org/terms

provide their maximum performance with a great amount of data, and at this point they take advantage of their highest parallelism. In high definition resolutions it seems to reach the OpenACC limit, where the increase of its speedup levels off.

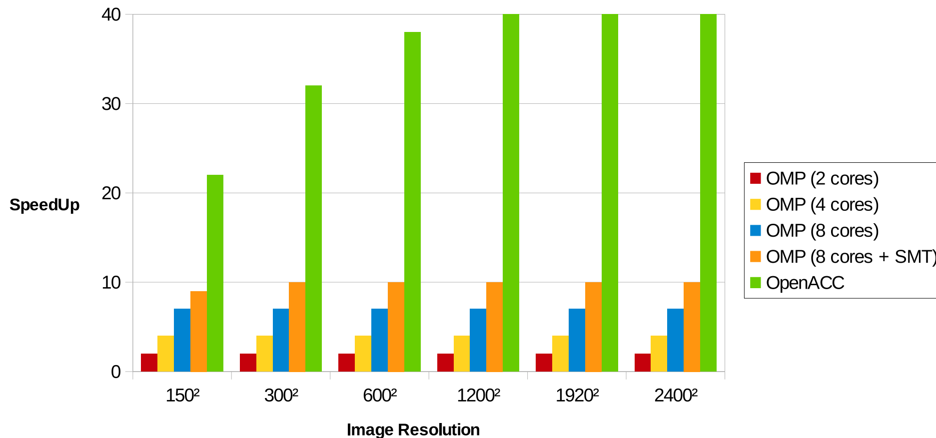

Figure 4. Speedup with different resolutions for DivergingTree using filter=5 and block=15 configuration.

We would like to emphasize that the use of a GPU programmed by applying the OpenACC paradigm not only makes it possible to obtain a satisfactory performance, but it also involves a programming task that is perfectly affordable. As an example, we can state that the total number of lines in the source code added or changed only represents 8% (5% of source code added and 3% modified).

5.5 Performance Analysis in other gradient models

In this subsection, we analyse the behaviour of the fists stages of other motion estimation models such as Multi-channel Gradient Model (McGM)[16,17] which is also based on temporal and spatial filtering stages. Due to the analogy in the filtering operation in both models, this analysis allows to extrapolate the maximum achievable throughput using based-directive coding in more complex algorithms.

In particular, McGM is based on FIR-Filtering Temporal stage where three different temporal channels are computed according to the experiments carried out by Hess and Snowden[18] about visual channels discovered in human beings: one low-pass filter and two band-pass filters with a center frequency of 10 Hz and 18 Hz, respectively. The FIR-Spatial Derivatives stage is based on space domain computation where the shape of the receptive fields from the primitive visual cortex is modelled using either Gabor functions or a derivative set of Gaussians.[19]

Figure 5 shows the speedups achieved using OpenACC programming paradigm in the first stage of the algorithm McGM. Bars correspond to the different programming models considered, directive-based models as OpenACC supported by CAPS and PGI compilers, and hand coding by means of CUDA. Figure 5 shows the performance rates varying images sizes (from 32^2 to 256^2) and different temporal filter sizes (L=7, 11, 15 and 19). The speedups are satisfactory for all cases considered. As expected, higher resolution images or larger filter sizes report higher speedups due to the higher degree of parallelism that can be exploited. GPU exploitation based on directive programming is advisable in terms of speedups, reporting ratios close to the performance peak only achievable by means of hand-tuned codified by means of CUDA. We would like to remark that PGI executable is the most efficient based on directives compiler.

Figure 6 shows the performance rates observed in the spacial filtering stage of McGM algorithm. Although OpenACC programming reports acceptable gains, the accelerations are substantially lower than those achieved at the stage of temporal filtering. Version developed in CUDA produces again the best performance figures because it is based on the convolution that efficiently exploit GPU's shared and constant memory.[20] Performance drop in the CAPS and PGI counterparts respect CUDA is due both compilers are not able to exploit efficiently GPU

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 07/16/2015 Terms of Use: http://spiedl.org/terms

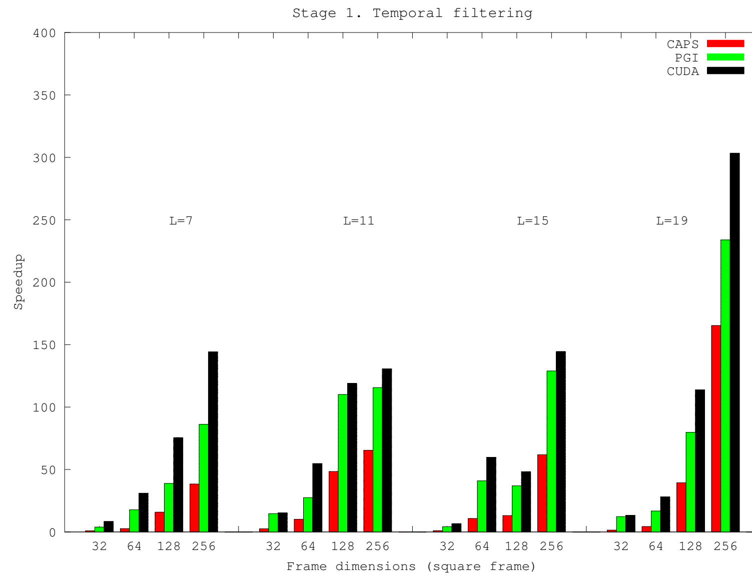

Figure 5. Temporal Filtering speedups with different resolutions varying temporal filter size (L).

memory hierarchy. However, speedups observed outperform 30× for configurations of spacial filter size of $T = 31$ in 256^2 images.

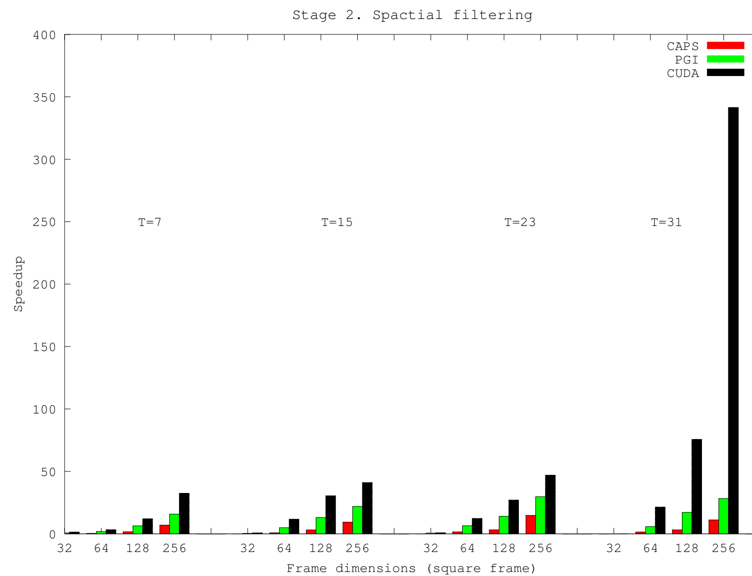

Figure 6. Spacial Filtering speedups with varying spatial filter size (T).

6. CONCLUSIONS

The results observed using a directive-based language for GPUs determined that the use of accelerators is an efficient solution not only in terms of performance but also in power consumption (watts/fps).

This work initially evaluates the performance gains for convolution operator using the programming paradigm OpenACC. The gains achieved are impressive. Using this first approach, GPU implementation using the directive paradigm of a optical flow algorithm also reports satisfactory performance rates.

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 07/16/2015 Terms of Use: http://spiedl.org/terms

The time required to estimate motion increases with the quality and resolution of the input image. The trend is to have higher and higher resolutions in multimedia environments, as we can appreciate in the new 4K resolution, which quadruples the resolution of the current FullHD video coding format (1920x1080).

Because of the need to achieve greater efficiency in implementations and due to the intrinsic parallelism inherent in the images to be exploited by the accelerator, we have used an NVIDIA card for our heterogeneous system.

One of the main potential handicaps to the wider acceptance of the use of accelerators by unskilled programmers is the know-how of the specific architectures involved, which implies a long learning curve. For this reason, the directive programming models such as standard OpenMP and the more recent OpenACC have been designed to exploit the accelerator independently of the architecture.

After evaluating our results, we can state that OpenACC is a promising alternative to consider, as it is affordable in terms of the learning required and it is possible to migrate C code in a straightforward way without making too many modifications (modifying and adding only 8% of code lines from their C source, which is negligible).

Finally, from a performance perspective, we can summarize as follows:

- The non-parallel version only provides a real-time response for a 150x150 resolution.
- The OpenMP version of a system based on 8 cores+SMT provides a real time response for 600x600 resolutions, which means a top speedup of 10×.
- The implementation based on the OpenACC paradigm is the only implementation which achieves real-time rates for 1200x1200 and close to the FullHD resolution (1920x1080), which again suggests that it is a serious option to consider. It achieves accelerations of upto 40× with respect to its serial CPU counterpart. Moreover, it offers improvements of between $3-4\times$ with respect to the best multicore version.
- Analyzing the behavior of another motion estimation algorithm as McGM, the first two stages based on temporal and spatial filtering also good performance are observed:
  - PGI version is the most successfully implementation based on directives programming.
  - Performance drop in the CAPS and PGI counterparts respect CUDA is due to the compiler is not able to exploit efficiently GPU memory hierarchy.

6.1 Acknowledgments

This work has been supported by Spanish Project TIN2012-32180 and partially supported by CAPAP-H4 Network (TIN2011-15734-E).

REFERENCES

[1] Chalmers, D. J., [*Conscious Experience*], Imprint Academic (1995).

[2] Bruce, V., Georgeson, M. A., Green, P. R., and Georgeson, M. A., [*Visual Perception: Physiology, Psychology and Ecology*], Psychology Press, 3 ed. (Sept. 1996).

[3] Zhaoping, L., [*Understanding Vision Theory, Models, and Data*], Oxford (2014).

[4] Anderson, A. J., Bruni, E., Bordignon, U., Poesio, M., and Baroni, M., "Of words, eyes and brains: Correlating image-based distributional semantic models with neural representations of concepts.," in [*EMNLP*], 1960–1970, ACL (2013).

[5] Behrens, S., Laue, H., Althaus, M., Bhler, T., Kuemmerlen, B., Hahn, H. K., and Peitgen, H.-O., "Computer assistance for mr based diagnosis of breast cancer: Present and future challenges.," *Comp. Med. Imag. and Graph.* **31**(4-5), 236–247 (2007).

[6] Garcia, C., Botella, G., Ayuso, F., Gonzalez, D., Prieto-Matias, M., and Tirado, F., "Implementation of a low-cost mobile devices to support medical diagnosis.," *Comp. Math. Methods in Medicine* **2013** (2013).

[7] Horn, B. K. P. and Schunck, B. G., "Determining optical flow," *Artificial Inteligence* **17**, 185–203 (1981).

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 07/16/2015 Terms of Use: http://spiedl.org/terms

[8] "CUDA: Compute Unified Device Architecture.." NVIDIA Corporation. (2007).
[9] "OpenACC Application Programming Interface v2.0." OpenACC Standard (2013).
[10] Koenderink, J. J., "Optic flow," *Vision research* **26**(1), 161–179 (1986).
[11] Nakayama, K., "Biological image motion processing: a review.," *Vision research* **25**(5), 625–660 (1985).
[12] Baker, S., Gross, R., and Matthews, I., "Lucas-kanade 20 years on: A unifying framework: Part 3," *International Journal of Computer Vision* **56**, 221–255 (2002).
[13] Lucas, B. D. and Kanade, T., "An iterative image registration technique with an application to stereo vision," in [*Proc. of 7th Int. Joint Conf. on Artificial Intelligence (IJCAI '81)*], 674–679 (April 1981).
[14] Barron, J. L., Fleet, D. J., and Beauchemin, S. S., "Performance of optical flow techniques," *International Journal of Computer Vision* **12**, 43–77 (1994).
[15] Fleet, D. J., [*Measurement of Image Velocity*], Kluwer Academic Publishers, Norwell, MA, USA (1992).
[16] P. W. McOwan, C. P. B. and Johnston, A., "Robust velocity computation from a biologically motivated model of motion perception," *Proceedings of the Royal Society B* **266**, 509–518 (1999).
[17] Liang, X., McOwan, P. W., and Johnston, A., "Biologically inspired framework for spatial and spectral velocity estimations," *J. Opt. Soc. Am. A* **28**, 713–723 (Apr 2011).
[18] Snowden, R. J. and Hess, R. F., "Temporal frequency filters in the human peripheral visual field.," *Vision Res* **32**(1), 61–72 (1992).
[19] J.J, K., "Optic flow," *Vision Res* **26**, 161–180 (1996).
[20] Ayuso, F., Botella, G., Garca, C., Prieto, M., and Tirado, F., "Gpu-based acceleration of bio-inspired motion estimation model.," *Concurrency and Computation: Practice and Experience* **25**(8), 1037–1056 (2013).

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 07/16/2015 Terms of Use: http://spiedl.org/terms