# A Comparative Study of Implementation Strategies for Real-Time Video Processing

Pablo Odorico[1,2] Tomás Touceda[1] Claudio Delrieux[3,4]

[1] Departamento de Ciencias e Ingeniería de la Computación
[2] (corresponding author: pablo.odorico@uns.edu.ar)
[3] Departmento de Ingeniería Eléctrica y de Computadoras
[4] IIIE - CONICET, parcialmente financiado por SECyT-UNS
Universidad Nacional del Sur - Alem 1253 - (8000) Bahía Blanca - Argentina

**Abstract**

We present a comparative study of the efficiency and effectiveness of different implementation strategies for real-time video processing. In particular, we tested the performance of GPU processors (using the CUDA library) against the performance of quad-core PC Intel processors (using the Intel Performance Primitives library).

The test consisted on applying a standard group of image processing algorithms, including histogram equalization, and convolution filtering, to a number of high definition video sequences and measuring the performance of each implementation. The results show that GPU processing is extremely cost-effective, but with the drawback that the underlying programming framework is very architecture-dependent, making it prone to the hardware idiosyncrasies, and therefore less abstract and reusable.

**Keywords**: Video Processing, GPGPU, Many-Core Computing.

## 1 Introduction

Graphic processing units (GPUs) emerged aproximately ten years ago, fueled by the widespread interest in computer games, 3D animation, after-effects production, and many other applications of computer graphics in the entertainment industry. GPU products have experienced since then a skyrocketing evolution in performance and capabilities, nearly doubling Moore's law [11]. Off the shelf

graphic cards are now able to peak over a teraflop processing capabilities, that is, several times more computing power at a fraction of the cost of the main processor. This trend was very soon recognised in the HPC community, and methods to take advantage of this huge computing power were foreseen in general purpose applications. This gave raise to GPGPU (general purpose GPU) [9].

Earlier use of GPUs in generic applications was based on creative ways of taking advantage of the rendering pipeline hardwired in the graphic card. In other words, a given computational problem, for instance huge matrices multiplication, was transformed and presented to the GPU in a way such that the rendering pipeline was indeed computing the desired result. For that reason, GPGPU programming was an extremely artisanal, hardware-specific task, with little or no abstraction, and therefore featuring neither code reuse nor portability.

Soon the GPU developers acknowledged this problem, and designed GPGPU programming platforms with a relatively higher level of abstraction, being NVIDIA's CUDA probably the most remarkable and widespread example (see for instance `http://www.nvidia.com/cuda`). As of today, NVIDIA's technologies claim to achieve supercomputing performance in a desktop system, at very reasonable costs [13].

On the other hand, CPU makers have developed libraries that take advantage of the available hardware features to the limit, including the increasing amount of cores, and the availability of SIMD operations. An outstanding example of this trend is the recent release of Intel's Integrated Performance Primitives (IPP), that usually provides image processing capabilities more than an order of magnitude faster that what could be achieved through a high level platform including optimizing compilers.

One of the most important aspects to consider in the development of HPC applications is the tradeoff between the cost-effectiveness (the acceleration provided under similar costs) and the programming flexibility (the ability to represent and use programming abstractions that allow the use of software engineering methodologies). There are several HPC architectures in the literature (multicore computing [1], grid computing [2], cluster [6], among others), each with a specific set of advantages over the others. However, there are few comparative studies of the performance, cost-effectiveness, and programming flexibility of these architectures in specific applications or contexts. It is unlikely that any of these technologies will be definitely shown to be superior to others in general purpose applications, and therefore, given a specific problem in hand, the most sensible way to apply an HPC schema is to use an appropriate benchmarking.

This work is aimed to provide comparative results of the cost-effectiveness, and programming flexibility of some implementation strategies in real-time video programming. In particular, we tested the performance of GPU processors

against quad-core PC Intel processors (using IPP). The processing benchmark implemented several general purpose image and video processing techniques, including histogram equalization, morphology, and convolution filtering. Similar processing techniques were applied to high definition video files.

In the next Section we describe the different implementation strategies that were used in this work, and the video processing algorithms that were applied. In Section 3 we show the implementation details, whose results are presented in Section 4. Finally, in Section 5 we discuss the conclusions and directions for further work.

## 2 Real-Time Image and Video Processing

Most image and video processing problems may be tackled using a relatively small set of basic techniques, whose usefulness, properties, and implementation and optimization details were thoroughfully studied in the last decades under the generic subject of *image processing* [3, 5, 8]. Among those fundamental techniques we may mention histogram manipulation and binarization; convolution, morphology, and other local, mask-based processing; over- and undersampling, including interpolation, rotation, and reconstruction techniques; quantization, including luminance, color, and chromatic-space manipulation algorithms; and spectral processing, including Fourier and DCT transform [4, 7, 10].

Even though the fields of video processing and computer vision require time-specific processing techniques, most of them are explicitly or implicitly founded on the basic image manipulation algorithms mentioned above, or are just extensions of the 2D versions of the algorithms to the 2D plus time domain [12, 14]. Therefore, the performance of any real-time application including digital video will depend highly on the implementation efficiency of these basic set of processing operations.

For these applications, a direct implementation of these algorithms in high level programming platforms so far exeeds the capabilities of standard PC computers, since even the most aggressive optimizing compilers cannot meet the standards and requirements of real-time, high definition video processing. This, however, does not mean that desktop computers are lacking computational power. Instead, the problem is to devise implementation strategies that circumvent the bottlenecks produced by serial code programming platforms in modern parallel, multicore hardware. In domain-specific areas, CPU makers released libraries that take advantage of the available hardware features (for instance IPP) in a way such that facilitates a rapid development of high-level applications with very high execution performance. Other implementation strategy, that appears to be increasingly fruitful in the embarrassingly parallel domain of image and video processing, is to take advantage of the huge amount of cores present in

Figure 1: Sample frames of the four HDTV video sequences used for testing purposes.

the GPU hardware, developing the processing operations in a GPGPU programming platform. These two are the implementation strategies that we tested and benchmarked in this work.

We have developed a testing framework, in which a meaningful subset of the basic image processing operations were implemented both on the CPU and the GPU. Such operations commonly take part in more complex processing pipelines. We used four high definition video sequences (1920x1080) from the SVT High Definition Multi Format Test Set, encoded with lossless compression (see Fig. 1). This test set has been specifically designed to evaluate the performance of video processing technologies. We have chosen to work with high definition video as it is becoming the standard resolution for television and internet broadcasting. The video sequences were obtained from the National Institute of Standards and Technology (NIST) (ftp://vqeg.its.bldrdoc.gov/HDTV/SVT_MultiFormat/SVT_MultiFormat_v10.pdf ).

# 3 Implementation Details

For the CPU-based implementation we have used the library IPP. The main reason for using IPP is the high level of optimization that it provides for Intel's microprocessors on an algorithmic level, and on the extensive use of the microprocessor's technologies, such as SSE in it's different versions. All the test operations developed for the GPU were implemented for NVIDIA's hardware, using the CUDA Toolkit and SDK version 3. A detailed explanation of some of the processing functions follows.

**Convolution** Implemented with support for 3x3 non separable kernels, and applied to each RGB channel individually.

**Morphology** 3x3 dilation and erosion applied to the luminance channel.

**Histogram equalization** A 256-bucket histogram is computed using a 1 byte representation of the luminance channel of the frame. A cumulative frequency function is calculated on the CPU, and applied back to the original frame in the GPU.

**Local luminance manipulation** Typical brightness adjustment functions (gamma correction, contrast enhancement, and so on) are computed modifying pixelwise the luminance channel.

In every operation, the individual frames have been uploaded to the GPU's global memory as a buffer in ARGB format (32 bits per pixel), then mapped to a texture so the pixels can be accessed with cached texture fetches. Since the pixel's format doesn't include luminance information, this is calculated on the fly for each pixel. We have used a two-dimensional execution layout with 16x16 blocks and one thread per pixel, which maximises the occupancy of the GPU processors. In the implementation of histogram equalisation, the GPU internal shared memory was used in order to reduce the number of uncached global memory fetches. The tests were performed on two computers, both running 64-bit Linux:

| System 1 | System 2 |
|---|---|
| Intel Core 2 Quad Q6600 @ 2.4 Ghz | Intel Core i7 920 @ 2.6 Ghz |
| 4GB DDR2 | 6GB DDR3 |
| XFX NVIDIA 9800GT | XFX NVIDIA GTX295 (only one chip) |
| Asus P5N32-E SLI | Asus P6T7 WS Supercomputer |

With respect to the performance evaluation method, for each video sequence the following procedure is performed:

1. A large number of frames are loaded into the system's RAM memory. This prevents the hard drive from becoming a bottleneck.

Figure 2: A frame and the processing result after border detection, and a pipe of operations.

2. For each operation:

   For the CPU process, the operation is applied to all the frames in the sequence, and these are later saved in another memory location. The elapsed time is evaluated, and the average processing time of a single frame is estimated as the ratio of the elapsed time divided by the number of frames in the sequence.

   For the GPU process, the timers provided by the GPU were used to accurately measure the processing time of each individual frame. In this case, the uploading and downloading time can also be taken into account.

# 4 Results and Discussion

A pipeline of operations was applied to the four video sequences mentioned in Section 2. In Fig. 2 we show a region of an original frame and the processing result thereof. The processing time was very similar in the four video sequences. In Fig. 3 we compare the individual processing time for every operation under the different implementation strategies. Finally, in Fig. 4 we show the execution time per operation of a processing pipeline in CPU vs. GPU. All the tests were performed using HDTV format (1920x1080).

The results show that, in the context of digital image and video processing, the GPUs performs several times faster than the CPUs. This figure doesn't

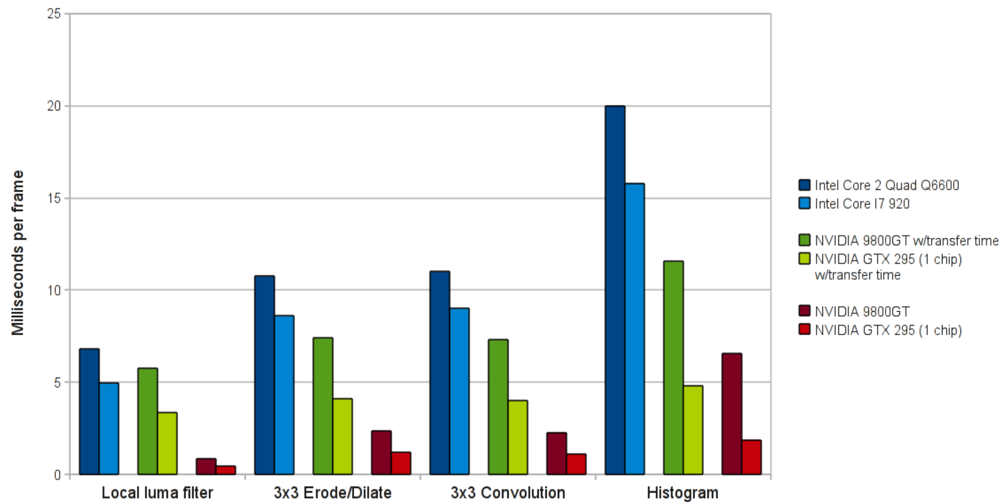


Figure 3: Processing time of the operations under different implementation strategies.

even take into account the fact that our GPU implementation is not as tightly optimized as IPP. In particular, neither of these CPUs is able to perform real-time processing since the required processing time is above 33ms (30fps). This –together with the fact that several GPUs can be installed to work in parallel, with a roughly linear speedup rate– shows the huge potential for GPGPU in real-time video applications. Also, given the retail cost, the cost-effectiveness is clearly better in GPU than in CPU. The comparison among two successive generations of both technologies shows that this tendency is even increasing, meaning that the competitive advantages of GPU over CPU in this type of applications is not merely circumstantial, and will certainly be huger in the future.

# 5 Conclusion

In this work we developed a testing framework that included a meaningful subset of the basic image processing operations. These operations were implemented both on CPU and GPU programming pipelines. We used four HDTV video sequences (1920x1080) from the SVT High Definition Multi Format Test Set, encoded with lossless compression. The results show a very clear advantage to GPU, both in execution time as in cost-effectiveness.

On the other hand, the lack of abstraction mechanisms in the GPGPU programming platforms makes almost impossible to develop reasonably complex

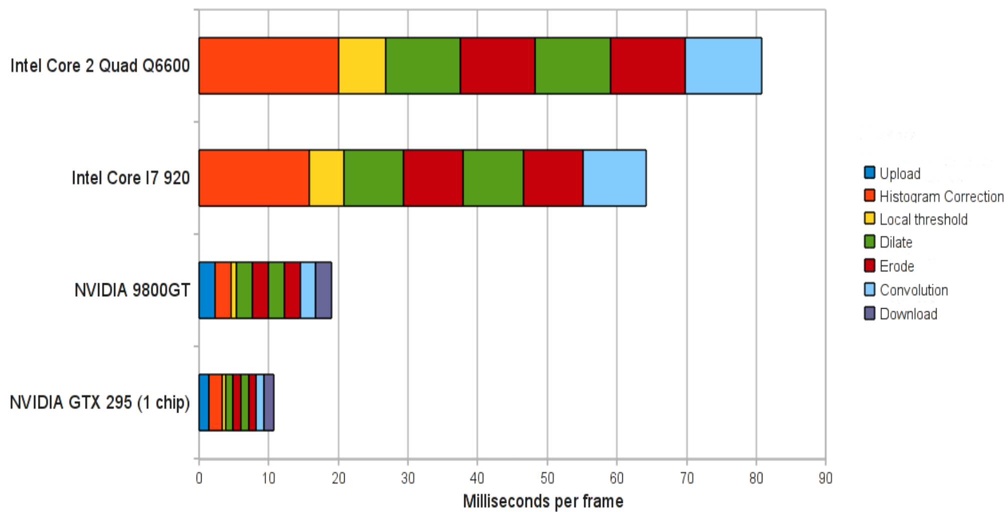


Figure 4: Execution time per operation of a processing pipeline in CPU vs. GPU.

tasks without a detailed exposure of the execution model. Therefore, computer-intensive video applications will be hybrid in nature, progressively relying on GPU power for easily programmable tasks, and relieving the CPU load which can be applied to other tasks, such as encoding and streaming operations.

The natural evolution of GPU programming frameworks will eventually reach more reasonable levels of abstraction. For instance, the almost-to-be-released Fermi platform promises full C++ facilities, which will strongly facilitate code development, reuse, and portability. However, this does not imply that development productivity in massively parallel applications will increase smoothly, since finding adequate algorithmic implementations in this context is not only a matter of programming abstractions, it also requires a radically new way of developing. This is probably going to be the main unavoidable paradigm shift in the future of programming, since CPU architecture innovation almost reached the theoretical limit for serial acceleration, and the only possible venue appears to be many core, multithread development.

# References

[1] Shameem Akhter and Jason Roberts. *Multi-Core Programming: Increasing Performance through Software Multi-threading*. Intel, 2006.

[2] Kentaro Fukui Bart Jacob, Michael Brown and Nihar Trivedi. *Introduction to Grid Computing*. IBM RedBooks Publishing, USA, 2005.

[3] Gregory A. Baxes. *Digital Image Processing*. John Wiley and Sons, New York, 1994.

[4] K. Castleman. *Digital Image Processing*. Prentice-Hall, New York, 1989.

[5] B. Dougharte and P. Giardino. *Matrix Structured Image Processing*. Prentice-Hall, Cambridge, MA, 1991.

[6] Rajkumar Buyya (editor). *High Performance Cluster Computing: Architectures and Systems,*. Prentice Hall, NJ, 1999.

[7] Andrew Glassner. *Principles of Digital Image Synthesis*. Morgan Kaufman, San Francisco, 1995.

[8] Rafael González and Richard Woods. *Digital Image Processing*. Addison-Wesley, Wilmington, USA, 1996.

[9] Mark Harris and Kavid Luebke. GPGPU Tutorial. Supercomputing 2006 Conference, 2006.

[10] Anil Jain. *Fundamentals of Digital Image Processing*. Prentice-Hall, Cambridge, 1996.

[11] Kenny Yeo. Voodoo Beginnings - 10 Years of GPU Development. *PcStats*, 2009.

[12] M.Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision*. PWS Publishing, Pacific Grove, CA, 1998.

[13] Gale Reference Team. Nvidia tesla gpu processor ushers in personal supercomputing. *Mainframe Computing Newsletter*, 20(8), 2007.

[14] S. E. Umbaugh. *Computer Vision and Image Processing*. Prentice-Hall, New York, 1998.