Application of Multi-core and GPU Architectures on Signal Processing: Case Studies

Alberto Gonzalez[1], José A. Belloch[1], Gema Piñero[1], Jorge Lorente[1], Miguel Ferrer[1], Sandra Roger[1], Carles Roig[1], Francisco J. Martínez[1], María de Diego[1], Pedro Alonso[2], Víctor M. García[2], Enrique S. Quintana-Ortí[3], Alfredo Remón[3] and Antonio M.Vidal[2]
Correspondence author: agonzal@dcom.upv.es
Audio and Communications Signal Processing Group [1] (GTAC) iTEAM, Universidad Politécnica de Valencia
Department of Information Systems and Computation[2] (DSIC) Universidad Politécnica de Valencia
Department of Computer Science and Engineering[3] (ICC) Universidad Jaume I de Castellón

Abstract

In this article part of the techniques and developments we are carrying out within the INCO2 group are reported. Results follow the interdisciplinary approach with which we tackle signal processing applications. Chosen case studies show different stages of development: We present algorithms already completed which are being used in practical applications as well as new ideas that may represent a starting point, and which are expected to deliver good results in a short and medium term.

Keywords: Multi-core/GPU Architectures, Structured linear systems, FFT, Convolution, MIMO detection, LDPC codes, Array processing, Adaptive algorithms.

1. Introduction

INCO2 [1] is a group of excellence in the Comunidad Valenciana (Spain), recognized as such by the local government through the PROMETEO 2009/013 project award. The members of the INCO2 group address problems arising in Signal Processing applications from an interdisciplinary perspective, designing solutions based on high performance hardware and developing algorithmic techniques with a modern and advanced software conception. In [2], both the architectural design and programming models of current general-purpose multi-core processors and graphics processors (GPU) were covered, with the goal of identifying their possibilities and impact on signal processing applications. Probably, the best form of appreciating the effect of these new architectures on signal processing is to analyze the performance attained by multi-core/GPUs architectures in the solution of a variety of applications. As a natural continuation of that work, in this paper we present several case studies that show how parallelization on multi-core/many-core architectures can be applied to specific problems and the outcome of it.

The rest of the paper is organized as follows. In Section 2 we show how to parallelize a detection method for MIMO digital communications systems on multi-core architectures. An evaluation of several packages to compute the FFT is presented in Section 3. Section 4 is devoted to the solution of Toeplitz linear systems on GPU and the parallelization of a beamforming algorithm for microphone arrays in Section 5. In Section 6 adaptive algorithms in digital signal processing systems with parallel convex combinations are presented. We dedicate Section 7 to present two potencial applications to be developed in GPU in the near future by INCO2: Multichannel convolution and the decoding of LDPC codes. Finally some concluding remarks are reported in Section 8.

2. Direct search methods for MIMO Systems

An emerging technology for communication is transmitting through many input and output systems, which are known as MIMO systems [3]. This technology provides, among other advantages, an increase in the bandwidth and reliability of communications [4]. In this section, we will focus on the efficient detection of digital symbols transmitted through a MIMO system.

A wireless MIMO communication can be modeled by a system composed of M transmitting antennas and N receiving antennas. A complex signal $\mathbf{s}=[s_0,\ldots s_{M-1}]^T$, $\mathbf{s} \in C^M$ is sent, where the real and imaginary parts of each component belong to a discrete and finite set A (the constellation or alphabet), and a signal $\mathbf{x} \in C^N$ is received. Signal $\mathbf{x}$ is a linear combination of the transmitted signal $\mathbf{s}$, perturbed with additive white Gaussian noise $\mathbf{v} \in C^N$; therefore, $\mathbf{x}$ can be written as $\mathbf{x} = \mathbf{H}\ \mathbf{s} + \mathbf{v}$, where the entries of the NxM (channel) matrix $\mathbf{H}$ are complex. The optimal or maximum-likelihood (ML) detection of the sent signal means that, for each signal, the following discrete minimization problem must be solved: $\min_{\mathbf{s}} \|\mathbf{x}-\mathbf{H}\,\mathbf{s}\|_2$. Further details about MIMO detection can be found in [4].

When the dimensions of the problem and/or the size of the constellation grow, the computation of the optimal solution becomes very expensive [5]. In response to this, many heuristic techniques have been examined as alternatives. Our research group has studied the application of parallel computing to the different existing solvers. An approach is to use standard discrete minimization algorithms and adapt them to the problem, such as the Direct Search methods, which were first described in [6] and more recently revisited in [7]. These methods can be parallelized with two different goals in mind; following a common practice, we could use parallelism to reduce the computing time; alternatively, it can also be used to increase the probability of obtaining the optimal (ML) solution. This can be achieved by performing several searches in parallel using different initial points. We have adapted these methods to the MIMO detection problem, first with sequential versions and later with parallel versions of the sequential algorithms [8].

One of the most popular techniques for MIMO decoding is the Sphere Decoding algorithm [9]. This algorithm restricts the search to a sphere centered in the received signal x and with a given radius; it can be described as a search in a tree of solutions. The parallelism in this case can be exploited by assigning different branches of the tree to different processors. Several versions of this algorithm have been parallelized by the authors, using different parallel schemes, and different technologies (OpenMP and a hybrid method) [10]. The different versions were tested

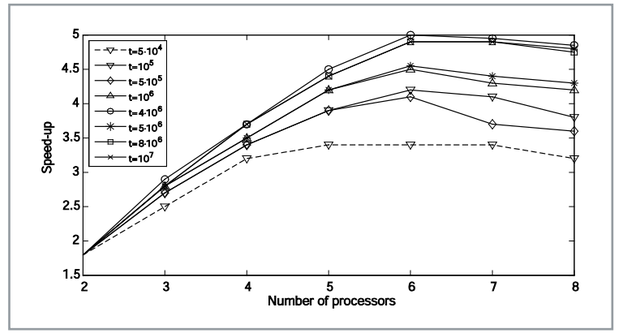


Figure 1. *Speed-up obtained using OpenMP.*

in a multi-core cluster composed of two PRIMERGY RXI600, each one with four Dual-Core Intel Itanium2 processors (1.4 GHz; 4 GB of shared RAM). The versions were tested with different problems, of increasing size (total number of nodes in the solution tree). The result is reported in terms of speed-up, which is the ratio between the time obtained with p processors and the best execution time obtained using a single processor. Figure 1 shows the speed-up attained with the parallel version based on OpenMP.

For all the problems tested, the best speedup is achieved with six processsors: compared with the time consumed by the serial version (one processor), the execution time is reduced by a factor of 5. Of course, these results strongly depend on the problem, and the results are comparatively better when the dimension of the problem is increased. Nevertheless, these results offer an idea of the possibilities of using parallel computing for this problem.

3. FFT on multi-core/many-core architectures

The discrete Fourier transform is one of the most important operations in Digital Signal Processing. Given a vector $\mathbf{x}=[x\ldots x_{n-1}]^T$ its DFT is defined as the matrix-vector product: $y_k = \sum_{j=0}^{n-1} w_n^{kj} x_j$, where $w_n = \exp\left(-2\pi i / n\right)$ and $i^2=-1$. The DFT can be used, among others, to obtain the frequency spectrum of a signal.

In many applications, the cost of computing the DFT [11] is too high; this is the case, e.g., of real-time applications. In those cases, the fast Fourier transform (FFT) can alleviate this problem of calculating the DFT. In particular, given a vector of size n, the computational cost of the DFT is $O(n^2)$ flops (floating-point arithmetic operations) while FFT requires only O(n log n) flops. In several experiments, we have evaluated some implementations of the FFT from different libraries on two different parallel architectures based on a multi-core processor and a GPU (see Table 1). Specifically, on the multi-core processor three libraries have been used: MKL (Intel), IPP (Intel)

Among the most widespread options in signal processing, the new multi-core / GPU architectures will be likely present in the next few years

Processors	*#cores*	*Frequency (GHz)*
Intel Xeon Quadcore E5405	8	2.33
NVIDIA Tesla C1060	240	1.3

Table 1. *Characteristics of the arquitectures used in the experiments.*

and FFTW (Massachusetts Institute of Technology). On the GPU, CUFFT (NVIDIA) and Volkov (an implementation coded by Vasily Volkov) have been evaluated, see Table 2.

The experiments comprised the computation of several FFT of a vector using single-precision arithmetic, with the size of the input vector varying from 8 to 8200 elements. The number of FFT computed in each experiment is proportional to the vector size, so the product between the vector size and the number of executions equals 8388608 (this number ensures more than 1000 executions with the biggest vector size used and is also a multiple of all the employed vector sizes). The performance (in terms of GFLOPS or 10^{15} flops per second) is computed using the same reference cost $5n\log_2 n$ for all experiments.

Library	*Version*	*Architecture*
FFTW3	3.2.1	Multi-core
Intel MKL	10.1	Multi-core
Intel IPP	6.0.2.076	Multi-core
NVIDIA CUFFT	2.3	GPU
Volkov		GPU

Table 2. *Libraries evaluated in the experiments.*

Figure 2 shows the performance obtained when the number of elements of the input vector is a power of two. As it can be seen, the performance of the kernels that operate on the GPU is notoriously higher than that of the multi-core counterparts. Other experiments were carried out for instance taking a prime number of elements of the input vector. In this case, all the FFT kernels suffered an important degradation, with the decrease being especially important for CUFFT, which yields the lowest-performance.

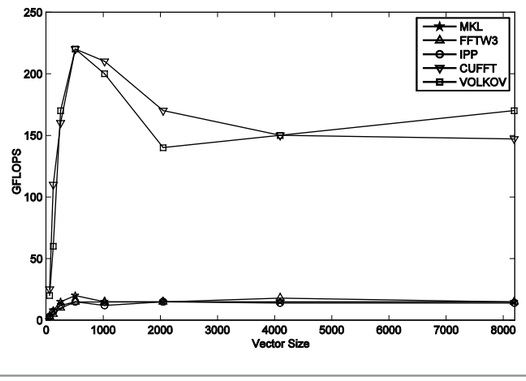


Figure 2. *Relation between GFlops and Vector Size when the number of elements of the input vector is a power of two.*

A preliminary conclusion from this study is that the FFT kernels in current libraries for the GPU clearly outperform those in libraries for the multi-core processors. However, much work remains to be done to fully optimize both types of kernels.

4. Solving structured systems on GPUs

Structured linear systems can be defined as:

$$\mathbf{TX=B} \qquad [1]$$

where $\mathbf{T}\in=\mathbb{R}_{nxn}$ is a structured matrix, $\mathbf{B}\in=\mathbb{R}_{nxn}$ *rhs* contains the right-hand side vector, and $\mathbf{X}\in=\mathbb{R}_{nxn}$ is the sought-after solution vector.

Some structured matrices, like Toeplitz, are characterized by an external structure (e.g., in the Toeplitz matrices all elements along diagonals are equal). Hankel and Vandermonde are also examples of structured matrices with an explicit external structure. The field of structured matrices also includes some classes with non-external structure, like the inverse of a Toeplitz matrix or Cauchy-like matrices. A formal definition of structured matrices is based on the property known as displacement structure [12], which basically sets that there exist one (symmetric case) or two (non-symmetric case) matrices of *nxr* (r<<n) containing the same information as an nxn structured matrix.

Structured matrices appear in a wide range of engineering applications as, i.e., digital signal processing. Other appealing feature of structured matrices deals with the existence of fast algorithms which allows to solve problem (1) with an order of magnitude lower than classical algorithms that does not exploit its special structure.

We will describe next our approach to solve problem (1) where $\mathbf{T}$ is a symmetric Toeplitz matrix using a GPU. To tackle this problem, we first searched for algorithms with an intrinsic parallelism, which allowed the use of a large number of threads working concurrently on the problem. One such an algorithm performs the triangular decomposition ($\mathbf{LDL^T}$, for $\mathbf{L}$ lower triangular and $\mathbf{D}$ diagonal) of symmetric Cauchy-like matrices. This algorithm works on a so called generator matrix $\mathbf{G}\in=\mathbb{R}_{nx2}$, with the property that, with only two columns, it contains all the information of a given symmetric Cauchy-like matrix. The following is a scheme of the algorithm:

```
for j=1,n
   for i=j+1, n
      Use ith row of G to compute li,j
entry.
   end for
end for
```

The outer loop processes the columns of $\mathbf{L}$ while the inner loop inspects the i-th row of the j-th column. All the entries of a given column j (inner loop) can be computed concurrently. This motivates the use of a linear array of blocks of threads to compute all these entries in parallel on a GPU.

The solution of (1) using the previous algorithm is carried out by transforming the Toeplitz matrix $\mathbf{T}$ into a Cauchy-like one. This is performed by means of the Discrete Sine Transform, an FFT-related transformation that is carried out first in the CPU. When applied to symmetric Toeplitz matrices, this transform exhibits the property that it results into two independent Cauchy-like matrices of order $n/2$. This benefit allows the solution of larger problems, by solving the two independent systems in turns, overcoming the memory limits of GPU (4GB). Furthermore, it also enables the use of two GPUs in the solution of a single symmetric Toeplitz problem.

Figure 3 shows the performance improvement obtained by using one GPU over the CPU to solve problem (1) with different numbers of independent vectors.

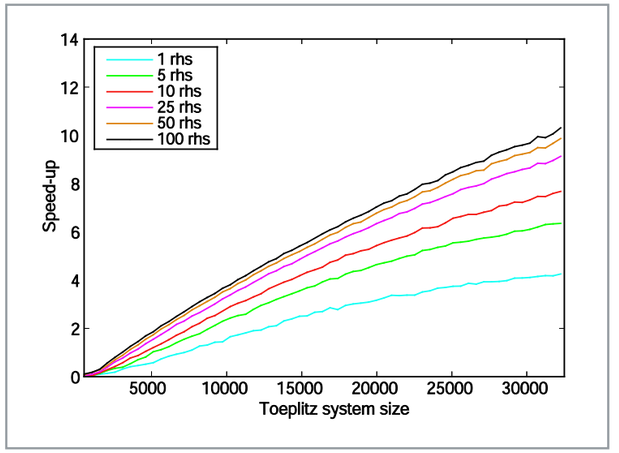


Figure 3. *GPU vs. CPU speed-up for a multiple right-hand side vectors systems.*

5. GPU array processing

A microphone array is a set of several microphones distributed in the space forming a specific pattern. Since a few decades ago, beamforming algorithms for microphone arrays have been studied and developed in order to improve the Signal-to-Noise Ratio (SNR) of the received signals, or to recover spatially separated signals considering their different Directions of Arrival (DoA) [13]. Nevertheless, one of their main limitations has been their high computational cost in practical acoustic environments where real-time sound processing must be carried out. Therefore we propose in this section an approach to the parallelization of some computations that are common to different beamforming designs.

System Model

Consider the system of Figure 4 where two loudspeakers are emitting two independent signals, $\mathbf{s}_1(k)$ and $\mathbf{s}_2(k)$, respectively, where k denotes the discrete time instant. At the other part of the room, three microphones are recording the mix of the two signals plus noise. This system can be modeled as a multichannel system with 2 inputs (loudspeakers) and 3 outputs (microphones), and the generalization to a multiple-input multiple-output (MIMO) system can be easily done.

Regarding the system of Figure 4, the problem is how to recover $\mathbf{s}_1(k)$ or $\mathbf{s}_2(k)$ by means of the signals recorded at the microphones. The approach taken herein makes use of signal processing algorithms to design the broadband beamformers (filters $\mathbf{g}_1$, $\mathbf{g}_2$ and $\mathbf{g}_1$ in Figure 4), once all the room channel responses ($\mathbf{h}_{nm}$ in Figure 4) are known. This problem is commonly known as signal deconvolution, and plays an important role in teleconferencing where the speech of interest has to be extracted from the observations of the microphone array but is usually corrupted by noise, room reverberation and other interfering sources.

According to Figure 4, the output of the n-th microphone is given by:

$$x_n(k) = \sum_{m=1}^{M}\sum_{j=1}^{Lh} h_{nm}(j)\, s_m(k-j) \qquad [3]$$

where n=1,2,…, N, being N the number of microphones and M the number of source signals, that is equal to the number of loudspeakers in Figure

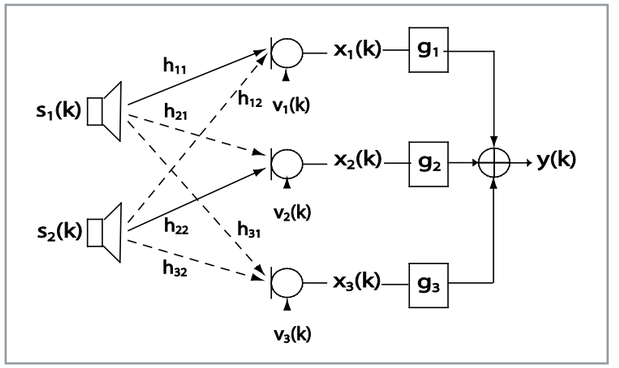


Figure 4. *System model for 2 loudspeakers (inputs) and 3 microphones (outputs).*

Structured matrices appear in a wide range of engineering applications as digital signal processing

4. L_h is the length of the longest room impulse response of all acustic channels $\mathbf{h}_{nm}$. The noise contribution has not been considered for sake of clarity.

This signal model can be rewritten in vector/matrix form as: $\mathbf{x}_n(k) = \mathbf{H}\mathbf{s}(k)$ where $\mathbf{x}_n(k)$ is a column vector and vector $\mathbf{s}(k)$ and matrix $\mathbf{H}$ are defined as $\mathbf{s}(k) = [\mathbf{s}_1^T(k) \quad \mathbf{s}_2^T(k)]^T$, where $\mathbf{s}_m(k)=[s_m(k)\ s_m(k-1)\ \ldots\ s_m(k-L_n+1)]^T$, and $\mathbf{H}=[\mathbf{H}_1\ \mathbf{H}_2\ \mathbf{H}_3]^T$, where $\mathbf{H}_n=[\mathbf{h}_{n1}^T\ \mathbf{h}_{n2}^T]^T$, and $\mathbf{h}_{nm}=[h_{nm,o}\ h_{nm,1}\ \ldots\ h_{nm,Lh-1}]^T$ for n=1,2,3 and m=1,2; $(\cdot)^T$ denotes the transpose of a vector or a matrix and L_h is the length of the longest channel impulse response. Once the recorded signals $\mathbf{x}_n(k)$ have been modeled, the broadband beamformers (filters $\mathbf{g}$) have to be designed and calculated. Benesty et al. [14] present an excellent state-of-the-art of the main algorithms used in audio applications. Some of them make use of the channel matrix $\mathbf{H}$ exclusively and calculate filters $\mathbf{g}_i$ based on its inverse (or pseudo-inverse), whereas other methods take also into account the correlation matrix of the recorded signals. Under perfect estimation of channel impulse responses, both types of filters show similar good performance, but in practical experiments, where the $\mathbf{h}_{nm}$'s are estimated under some uncertainties, filters based on the estimated correlation matrix outperforms those based on the channel inversion.

The estimated correlation matrix of spatially sampled signals, $\mathbf{x}_n(k)$, is commonly known in the literature as the Sample Correlation Matrix (SCM), and its expression is given by:

$$\hat{\mathbf{R}}_x = \frac{1}{N}\sum_{k=1}^{N}\mathbf{x}(k)\cdot\mathbf{x}^T(k) \qquad [4]$$

where $\mathbf{x}(k) = [\mathbf{x}_1^T(k) \quad \mathbf{x}_2^T(k) \quad \mathbf{x}_3^T(k)]^T$.

Regarding the dimensions of SCM, $[3L_g \times 3L_g]$, L_g depends on the length of the room impulse response L_h and is usually greater than 150. Considering that (3) has to be recalculated at short time intervals due to the non-stationary nature of sound signals, and that $N \geq 3L_g$ to assure that $\hat{\mathbf{R}}_x$ is full-rank and invertible, then an efficient parallelization of the computation in (4) is required. Three different implementations have been considered in order to obtain the matrix correlation as fast as possible. In one hand the sequential implementation and in the other hand two different parallel implementations: one in multiple cores of CPU and the other in the GPUs.

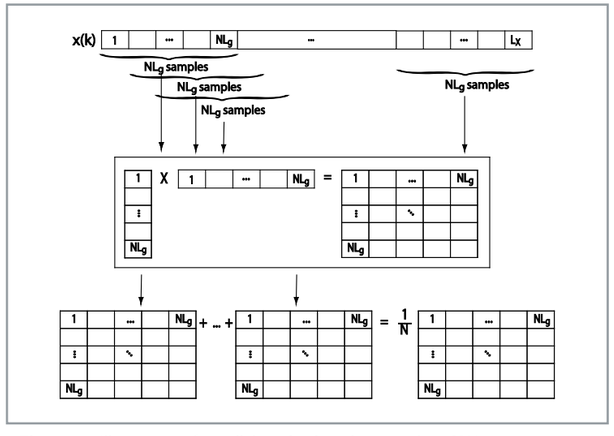


Figure 5. *Illustration of parallel method implemented on CPU.*

The sequential implementation

The sequential implementation of the Sample Correlation Matrix of (3) is iteratively done by a 'for' loop which calculates one vector outer multiplication and one matrix sum correspondent to index k-th of the total sum at each iteration. Its implementation can be seen schematically in Figure 5, where vector $\mathbf{x}(k)$ is split in smaller overlapping vectors of $3L_g$ length each.

Parallel implementations of the Sample Correlation Matrix of (3)

1) Parallelization in CPU multi-core:
In this case the parallelization consists in dividing the sequential tasks described above in different CPU cores. To achieve that the Matlab toolbox for parallel computing has been used, more specifically the functions *matlabpool* and *spmd* [15].

2) Parallelization on GPU:
In this case, the parallelization is performed at a lower level than in CPU. For this purpose, the software interface called Jacket [16], which allows running code in the GPU through Matlab, has been tested. The following steps have been taken:

- First, to send microphone array signals $\mathbf{x}(k)$ to the GPU using Jacket function *gdouble*().
- Second, to parallelize (3) so no iteration of 'for' loop must depend on a previous result. Then parallelization of the 'for' loop is done using the Jacket function *gfor*().

The step 2 has been carried out splitting each vector $\mathbf{x}_n(k)$ of (4) in basic blocks of variable length Z. The performed parallelization in GPU for $Z=L_g/2$ can be seen in Figure 6. Let us denote $\mathbf{x}_n^{(i)}(k)$ as the i-th block. Considering that $\mathbf{x}_n(k)$ has length NL_g, the number of available blocks $\mathbf{x}_n^{(i)}(k)$ is $NL_g/(L_g/2)=2N$. Therefore, the single outer product $\mathbf{x}_n(k)\mathbf{x}_n^T(k)$ of (4) is now computed in parallel at the GPU though $(2N)^2$ outer products $\mathbf{x}_n^{(i)}(k)\left(\mathbf{x}_n^{(i)}(k)\right)^T$.

Figure 6 shows the case for $N=3$ microphones, so there are $2N=6$ basic blocks available, and $\mathbf{x}_n(k)\mathbf{x}_n^T(k)$ will be computed whith $(2N)^2=36$ outer products in parallel.

		Lg=110	Lg=130	Lg=150	Lg=170	Lg=190
Sequential		0.6084	0.9121	1.2269	1.7909	2.1333
CPU parallel		0.3908	0.5823	0.7853	1.0628	1.6098
GPU parallel	**9 parts**	**0.1353**	**0.2135**	**0.2702**	0.3702	0.5166
	36 parts	0.1866	0.2303	0.2749	**0.3656**	**0.4232**
	81 parts	0.3012	0.3557	0.4016	0.4980	0.5819

		Lg=210	Lg=260	Lg=300	Lg=330	Lg=360
Sequential		2.5883	6.0940	9.1885	11.264	14.681
CPU parallel		1.9913	192.14	585.32	*overflow*	*overflow*
GPU parallel	**9 parts**	0.6714	*overflow*	*overflow*	*overflow*	*overflow*
	36 parts	**0.4808**	**0.7813**	**2.1767**	4.6818	6.8380
	81 parts	0.6642	0.9183	2.3748	**3.5623**	**4.9330**

Table 3 *Table of times used in calculating the Sample Correlation Matrix (SCM) of equation (4).*

Signal deconvolution plays an important role in teleconferencing where the speech of interest has to be extracted from the observations of the microphone array usually corrupted by noise

Three different lengths of basic blocks have been tested in GPU: $Z=L_g$, $Z=L_g/2$ and $Z=L_g/3$, which results in N^2, $(2N)^2$ and $(3N)^2$ outer products of block vectors $\mathbf{x}_n^{(i)}(k)$. For the system depicted in Figure 4 whith $N=3$, the different sizes of Z give a parallelization of 9, 36 and 81 independent outer products for step 2, respectively.

Testing Results

Sequential implementation and both parallel implementations explained above for 3 recorded signals $x_n(k)$ at sampling frequency of 11 kHz have been tested with an i7 CPU of Intel and a NVIDIA GPX285 GPU. Results obtained for signals of duration 4 seconds (44 *k*samples) can be seen in Table 3. The CPU parallel method has been carried out with 3 cores, it has been proved that for this kind of computation it was the best configuration. As we can see in Table 3, the parallel implementation with multiples cores of a CPU only obtain speed up greater than 1 when $L_g \leq 210$ comparing to sequential implementation, achieving almost double velocity in the best case of L_g=110. An explanation for this low performance may be that too much time is lost distributing tasks into the different cores of the CPU, whereas the code to be distributed consists only in a few lines. Moreover, results of Table 3 show that computational time grows exponentially when L_g exeeds L_g=210; we suppose that in this cases, length of filters g_i is very large and buffers memory of the cores collapses: big amounts of data are replicated in all buffers, and that makes such a significant time increase. For L_g>300, Matlab returns a memory error because there is not enough memory to allocate matrices with such big dimensions.

Regarding GPU implementation, Jacket performs with matrix of maximum 65.536 elements. As we see in figure 6, dimensions of SCM depends on L_g so working whithin GPU configuration divided in 9 parts, when L_g=260 dimensions of SCM exceeds 65.536 elements, so Jacket program returns a memory error. To solve this problem we divide the calculation of SCM in more number of parts, and as table 3 show, as L_g grows, the number of parts used in the calculation of the matrix correlation must be incremented to reach the best time results. Otherwise, if we took the most efficient case for each length of filters, L_g, it can be shown that the speed-up in all the cases is near to 4, which means a significant time saving. Same results of Table 3 are depicted in Figure 7, where the graph at right shows those methods whose computation times are below 2 seconds. It should be noted that GPU outperforms sequential and multi-core implementations in all cases.

Finally it should be noted that, considering the duration of the recorded signals, 4 seconds, a delay in calculating the matrix correlation of one to three tenth of a second is attainable for real-time applications.

6. Adaptive algorithms with parallel combinations on multi-core platforms

During last years, adaptive systems [17] have been the objective of many studies due to their

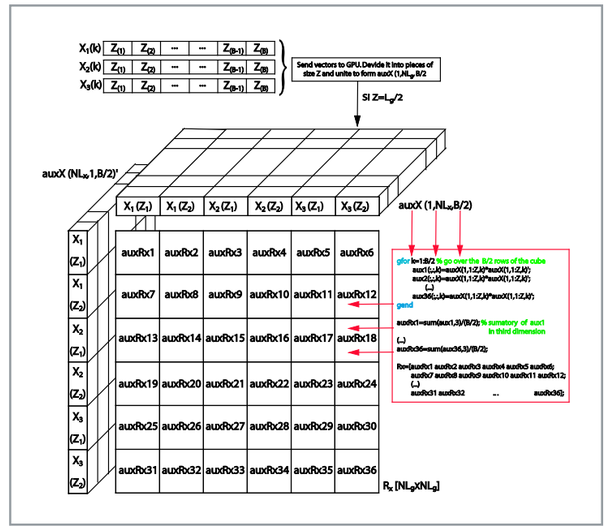


Figure 6. *Illustration of parallel method implemented on GPU.*

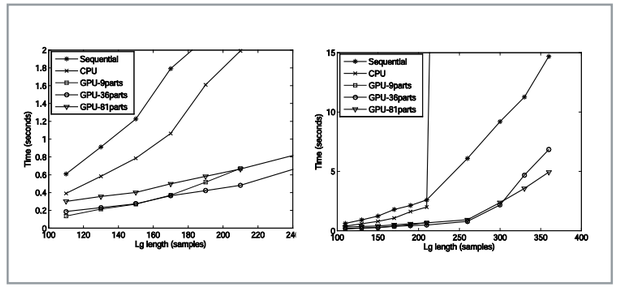


Figure 7. *Evolution of computational time when L_g grows.*

multiple applications in digital processing systems. Applications like channel identification, channel equalization or channel inversion, used for sound or communications systems, echo cancellation, noise cancellation, among others, are based on adaptive systems. There is a big amount of adaptive algorithms in order to control adaptive systems like: LMS, RLS, FTF, AP, etc. A complete description of each can be found in [18], whose conclusion says that none of them is globally better than the others, but also, the algorithms which achieve the best performances are the ones which have greater computational cost. Also, the ones which have good behavior in a permanent regime are worse than others if we compare the convergence speed. This is the reason why there are different adaptive strategies. In order to improve the performance of different adaptive algorithms, new parallel combining strategies have appeared, like convex [19]. These strategies allow to combine the strengths of two adaptive algorithms which present complementary features (for instance, one with fast convergence and the other with low residual error level in permanent regime), achieving the combination of the best performances from each one in a separated way. As it can be checked in [20], using this strategy is possible to achieve both objectives at the same time, high convergence speed and low residual error level in permanent regime. This kind of strategies can be used successfully in active noise control applications [21], obtaining a really good performance: fast convergence and low residual noise level. However, this better behavior appears at the expense of doubling the computational cost, since two algorithms have to be executed at the same time, in parallel. The parallel nature of this structure allows the distribution of the computation within parallel hardware like the multi-core systems, where the computational load can be easily dealed out among different cores and thus the execution time reduced. Therefore, the adaptive algorithm could be used in systems working at a higher sampling rate. The computations needed could be carried out in two kernels, using a third kernel to combine both algorithms, or using one of the first kernels to combine the signals if there are only two kernels. Next, Figure 8 shows the block diagram of the convex structure executed over a multi-core platform.

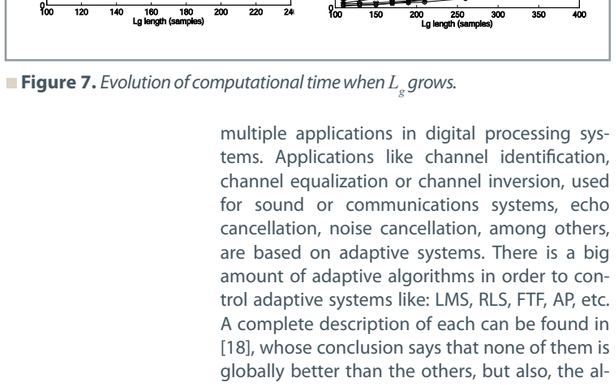


Figure 8. *Scheme of the multi-core convex combination.*

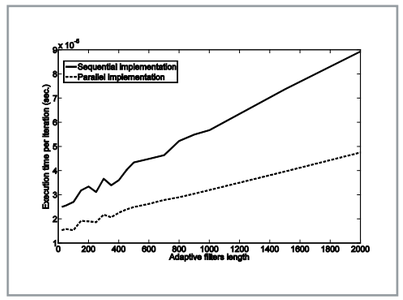


Figure 9. *Runtime per iteration for multi-core system and simple core system.*

As it can be checked in Figure 8, apart from the convex combinations, the rest can be executed in a parallel way. Therefore, the execution time has been reduced to the time that a single filters needs to carry out the computations. In other words, thank of this structure and the use of two kernels, the time required by the process becomes the time needed by one single kernel, instead of the double time required by a sequential implementation using monocore structures. Figure 9 exhibits the algorithm runtime per iteration and the comparison with the sequential version executed in a single core system. It shows the relation between the execution time and the length of the adaptive filters used in the convex structure when LMS algorithm is used as a controller of the adaptative filters. This test has been carried out on an Intel Core i7 CPU 920 @ 2.67GHz, and the algorithm was run in a Matlab R2009b platform using Parallel Computing Toolbox V4.2.

As it can be seen in Figure 9, the reduction of the algorithm runtime using a platform of two kernels is really significant. This structure only needs half runtime of the sequential one. The most important conclusion is that it will be possible to work with higher sampling frequencies in order to deal with signals with higher bandwidth, or just to have adaptive algorithms which require high computational load needing less time to carry out this operation.

7. Future Prospects

In this section, we present two potential applications in Signal Processing which are focused on the implementation of the multichannel convolution and the decoding of LDPC codes using the capabilities of GPU computing.

Multichannel convolution

It can be shown that the computation of the convolution operation consists of several scalar multiply and add operations [22], where a certain parallelism can be identified. In order to compute the convolution, the architecture of the GPUs allows different levels of parallelism. At a first level, a single convolution operation of two signals can be efficiently implemented in parallel inside a GPU. The second level of parallelism allows carrying out different convolutions of different channels parallelly. Note that, obviously, the benefits of using a GPU increase when both levels of parallelism are exploited simultaneously.

The possibilities that GPUs offer are varied. However, the main challenge when implementing an algorithm on GPU relies on adapting the resources of the GPU to obtain the best performance depending on the properties of the signals (monochannel, multichannel, etc.) and, of course, the type of processing that wants to be carried out: Convolution of all the signals either by the same $h(k)$ or with different $h_i(k)$ with $i \in [0 \ldots n\text{-}1]$, convolution of some signals by $h_1(k)$ and others by $h_2(k)$ and all of them at the same time, etc.

Recently, the new CUDA toolkit 3.0 lets use CUFFT [11] with the property concurrent copy and execution and therefore, implementing real-time applications where the latency of transfering the samples from the CPU to the GPU for processing and vice versa overlapped by computation.

LDPC Codes on GPU

Low-Density Parity-Check codes (LDPC codes) are linear block channel codes for error control coding with a sparse parity-check matrix (a matrix that contains few ones in comparison to the amount of zeroes). They have recently been adopted by several data communication standards such as DVB-S2 and WiMax. The concept of LDPC coding was first developed by Robert G. Gallager in his doctoral dissertation at the MIT in the begining of sixties [23] but quickly forgotten due to its impractical implementation at that moment and the introduction of Reed-Solomon codes. They were rediscovered by MacKay and Neal in 1996 [24].

These codes provide a performance very close to the Shannon capacity limit of the channel, low error floor, and linear time complexity for decoding (lower than turbocodes). We can find simple tutorials to understand the basics of these kind of codes in [25], [26], and software to test them in [27]. LDPC codes are inherently suited for parallel hardware and software implementations

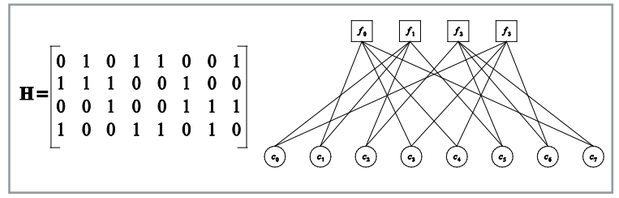


Figure 10. *Tanner graph of a linear block code parity-check matrix* $\mathbf{H}$.

as we can read in [28], [29] and [30] using CUDA and other GPU programming tools.

LDPC codes can be represented graphically by a Tanner graph [31] (an undirected bipartite graph with variable nodes, c_i, and check nodes, f_j). An example is shown in Figure 10, that corresponds with the parity-check matrix on its left $\mathbf{H}$:

LDPC decoders are based on variations of belief propagation, sum-product or message passing algorithms. In any of these algorithmic denominations, information flows to/from variable nodes and from/to check nodes until the algorithm converges to a stable state, finding the most likelihood transmitted codeword. An easy example can be observed in the bit flipping algorithm (hard decision decoding). The iterations are divided in two dependent steps:

1. Each variable node sends the majority voted bit to all its connected check nodes (at the beginning, the only information available is the received bit)
2. Each check node estimates each connected variable node bit as the parity-check matrix dictates (using the estimations of the rests of the connected variable nodes and excluding the value that is estimating) and send this information to this connected variable node.

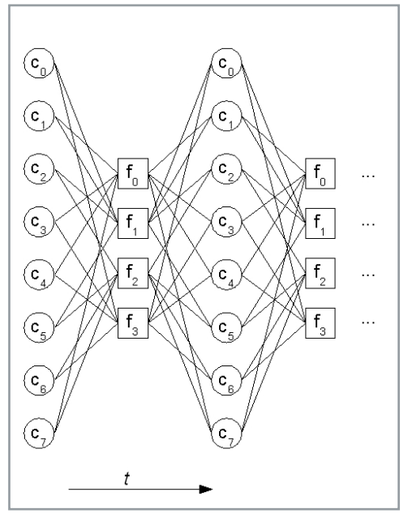


Figure 11. *Computation and message passing in parallel algorithm.*

LDPC codes are inherently suited for parallel hardware and software using CUDA and other GPU programming tools

These steps are executed iteratively until the estimated word is a codeword. Better results are obtained when soft decision is used [32]. It can be observed that the computations within the check nodes and within the variable nodes are alternated and interdependent in time, so they must be executed one after another because of their inherent sequentiallity. The computations in every check node are independent, so they are perfectly parallelizable; the same happens with the variable nodes computations. Within a check node, it must be computed a different result to every variable node that is connected to it. Something similar is observed regarding to the variable node computations. Figure 11 shows the dependency graph of the parallel algorithm.

We are focusing our implementations on concentrating the operations within every node because each result in a node shares nearly all the multiplication factors that it contains. Another important question is how the accesses to the global and shared memory are arranged in order to make a coalesced access and to avoid conflicts in the memory banks. This will ensure a good speedup in a real time environment.

9. Conclusions

Throughout this article it has become obvious the impact of new multi-core / GPU architectures in the field of signal processing. Among the most widespread options in signal processing, these new architectures will be likely present in the next few years. However, it is also very likely that FPGA devices keep a good share of the market, as they cover a large part of very specific applications.

The purpose of this work was to serve as a showcase of different signal processing applications in which new Multi-core/GPU architectures can be competitive. Different applications, in which researchers of INCO2 Group are working, have been used as case studies. The aim of this group is precisely the application of high performance computing and next-generation parallel architectures (particularly multi-cores and GPUs) in the solution of problems in signal processing. We believe this option is a sure bet in one of the most promising areas of current technology, in general, and the Information Technology area, in particular, where the duo computer-communications can not be dissociated.

Acknowledgements

This work was supported by Generalitat Valenciana Project PROMETEO/2009/013 and partially supported by Spanish Ministry of Science and Innovation through TIN2008-06570-C04 and TEC2009-13741 Projects.

References

[1] www.inco2.upv.es
[2] A. Gonzalez, J. A. Belloch, G. Piñero, F. J. Martínez, P. Alonso, V. M. García, E. S. Quintana-Ortí, A. Remón, and A. M.Vidal "The Impact of the Multi-core Revolution on Signal Processing"; Waves, vol. 2, 2010.
[3] A. J. Paulraj, D. A. Gore, R. U. Nabar, and H. Bölcskei, "An overview of MIMO communications - a key to Gigabit wireless," Proceedings of the IEEE, vol. 92, no. 2, pp. 198–218, Feb. 2004.
[4] S. Roger, F. Domene, C. Botella, G. Piñero, A. Gonzalez, and V. Almenar, "Recent advances in MIMO wireless systems", Waves, vol. 1, pp. 115-123, 2009.
[5] J. Fink, S. Roger, A. González, V. Almenar, and V. M. García, "Complexity Assessment of Sphere Decoding Methods for MIMO Detection", IEEE International Symposium on Signal Processing and Information Technology (ISSPIT), Ajman, UAE, December 2009.
[6] R. Hooke and T. A. Jeeves, "Direct Search solution of numerical and statistical problems", Journal of the Association for Computing Machinery, pp. 212–229, 1961.
[7] T. G. Kolda, R. M. Lewis, and V. Torczon, "Optimization by Direct Search: New perspective on some Clasical and Modern Methods", SIAM Review, vol. 3, pp. 385–442, 2003.
[8] R. A. Trujillo, A. M. Vidal, and V. M. García, "Decoding of signals from MIMO communication systems using Direct Search methods", 9th International Conference Computational and Mathematical Methods in Science and Engineering (CMMSE), Gijón, Spain, July 1-3 2009.
[9] M. Pohst, "On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications", ACM SIGSAM Bull., vol. 15, pp. 37–44, 1981.
[10] R. A. Trujillo, A. M. Vidal, V. M. García, and Alberto González, "Parallelization of Sphere-Decoding Methods," Lecture Notes in Computer Science, vol. 5336/2008, pp. 2-12, 2008.
[11] C. Van Loan, "Computational Frameworks for the Fast Fourier Transform," SIAM Press, Philadelphia, 1992.
[12] T. Kailath and Ali H. Sayed, "Displacement Structure: Theory and Applications", SIAM Review, vol. 37, pp. 297-386, Sept. 1995.
[13]H. Branstein and D.B. Ward, "Microphone Arrays: Signal Processing Techniques and Applications", Springer, Berlin (Germany), 2001.
[14] J. Benesty, J. Chen, Y. Huang, and J. Dmochowski, "On Microphone-Array Beamforming From a MIMO Acoustic Signal Processing Perspective", IEEE Trans. Audio, Speech, and Language Processing, vol.15, no.3, pp.1053-1065, 2007.
[15] Parallel Computing Toolbox 4.2 Product Page, The Mathworks, online at: www.mathworks.com/products/parallel-computing
[16] Jacket for MATLAB Product Page, AccelerEyes, online at: www.accelereyes.com/resources/literature.
[17] B. Widrow and S. D. Stearns, Adaptive Signal Processing, Prentice-Hall, Englewood Cliffs, N.J., 1985.
[18] S. Haykin, Adaptive Filter Theory, Prentice-Hall, Ed., Upper Saddle River, NJ, Fourth edition 2002.
[19] J. Arenas-García, A. R. Figueiras-Vidal, and Ali H. Sayed, "Mean-Square Performance of Convex Combination of two Adaptive Filters", IEEE Transactions on Signal Processing, vol 54, no. 3, March 2006.
[20] M. Ferrer, M. de Diego, A. González, and G. Piñero, "Convex combination of affine projection algorithms adaptive filters for ANC", 17th European Signal Processing Conference, Glasgow, Scotland, August 2009.
[21] M. Ferrer, M. de Diego, A. González, and G. Piñero, "Convex combination of adaptive filters for ANC", 16th International Congress on Sound and Vibration, Cracow, Poland, July 2009.
[22] S.S. Soliman, M. D. Srinath, "Continuos and discrete Signals and Systems", Ed. Prentice Hall.
[23] R.G. Gallager, "Low-Density Parity-Check Codes", MIT Press, Cambridge, MA (USA), 1963.
[24] D.J.C. MacKay and R.M. Neal, "Near Shannon limit performance of low density parity check codes", IEEE Electronics Letters, vol. 33, no. 6, pp. 457-458, 1997.
[25] M.J. Bernhard, "LDPC Codes -a brief Tutorial", users.tkk.fi/~pat/coding/essays/ldpc.pdf
[18] Overview of Low-density parity-check codes, online at: en.wikipedia.org/wiki/Low-density_parity-check_code.
[19] R. M. Neal, online at: www.cs.utoronto.ca/~radford/homepage.html
[20] G. Falcão, L. Sousa, and V. Silva, "Massive parallel LDPC decoding on GPU", Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Salt Lake City, Ut (USA), February 20 - 23, 2008.
[21] G. Falcão, V. Silva, and L. Sousa, "How GPUs can outperform ASICs for fast LDPC decoding", Proceedings of the 23rd International Conference on Supercomputing, Yorktown Heights, NY (USA), 2009.
[22] S. Cheng, "A Parallel Decoding Algorithm of LDPC codes using CUDA", tulsagrad.ou.edu/samuel_cheng/articles.html
[23] R. Tanner, "A recursive approach to low complexity codes", IEEE Transactions on Information Theory, vol. 27, no. 5, pp: 533-547, 1981.
[24] T. Richarson and R. Urbanke, Modern Coding Theory, Cambridge University Press 2008.

Biographies

Antonio M. Vidal
See page 73

Alberto Gonzalez
See page 74

Gema Piñero
was born in Madrid, Spain, in 1965. She received the Ms. in Telecommunication Engineering from the Universidad Politécnica de Madrid in 1990, and the Ph.D. degree from the Universidad Politecnica de Valencia in 1997, where she is currently working as an Associate Professor in digital signal processing. She has been involved in different research projects including active noise control, psychoacoustics, array signal processing and wireless communications in the Audio and Communications Signal Processing (GTAC) group of the Institute of Telecommunications and Multimedia Applications (iTEAM). Since 1999 she has led several projects on sound quality evaluation in the fields of automotive and toys. Since 2001 she has been involved in several projects on 3G wireless communications supported by the Spanish Government and Telefónica. She has also published more than 40 contributions in journals and conferences about signal processing and applied acoustics. Her current research interests in the communications field include array signal processing for wireless communications, MIMO multi-user techniques and optimization of signal processing algorithms for multi-core and GP-GPU computation.

Francisco José Martínez Zaldívar
See page 74

Pedro Alonso
See page 74

Alfredo Remón
See page 74

Víctor M. García
See page 74

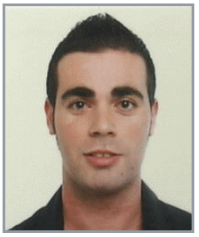
Enrique S. Quintana-Ortí
See page 75

Maria de Diego
was born in Valencia, Spain, in 1970. She received the Telecommunication Engineering degree from the Universidad Politecnica de Valencia (UPV) in 1994, and the Ph.D degree in 2003. Her dissertation was on active noise conformation of enclosed acoustic fields. She is currently working as Associate Professor in digital signal processing and communications. Dr. de Diego has been involved in different research projects including active noise control, fast adaptive filtering algorithms, sound quality evaluation, and 3-D sound reproduction, in the Institute of Telecommunications and Multimedia Applications (iTEAM) of Valencia. She has published more than 40 papers in journals and conferences about signal processing and applied acoustics. Her current research interest include multichannel signal processing and sound quality improvement.

Miguel Ferrer
was born in Almería, Spain. He received the Ingeniero de Telecomunicacion degree from the Universidad Politécnica de Valencia (UPV) in 2000, and the Ph.D degree in 2008. In 2000, he spent six months at the Institute of aplicated research of automobile in Tarragona (Spain) where he was involved in research on Active noise control applied into interior noise cars and subjective evaluation by means of psychoacoustics study. In 2001 he began to work in GTAC (Grupo de Tratamiento de Audio y Comunicaciones) that belongs to the Institute of Telecommunications and Multimedia Applications. He is currently working as assitan professor in digital signal processing in communications Department of UPV. His area of interests includes efficient adaptive algorithm and digital audio processing.

Sandra Roger
was born in Castellón, Spain, in 1983. She received the degree in Electrical Engineering from the Universidad Politécnica de Valencia, Spain, in 2007 and the MSc. degree in Telecommunication Technologies in 2008. Currently, she is a PhD grant holder from the Spanish Ministry of Science and Innovation under the FPU program and is pursuing her PhD degree in Electrical Engineering at the Institute of Telecommunications and Multimedia Applications (iTEAM). In 2009, she was a guest researcher at the Institute of Communications and Radio-Frequency Engineering of the Vienna University of Technology (Vienna, Austria) under the supervision of Prof. Gerald Matz. Her research interests include efficient data detection, soft demodulation and channel estimation for MIMO wireless systems.

José Antonio Belloch
See page 75



Jorge Lorente
was born in Algemesí, Spain in 1985. He received the Ingeniero Técnico de Telecomunicación degree from the Universidad Politécnica de Valencia, Spain, in 2007 and the MSc. degree in Telecommunication Technologies in 2010. Currently, he is working at the Institute of Telecommunication Technologies and Multimedia Applications (iTEAM). His research focuses on microphone-array beamforming algorithms and parallelization of signal processing problems on the different cores of a CPU and also on GPU.

Carles Roig
was born in Alginet, Spain, in 1986. He received the degree in Telecommunication Engineering from the Universidad Politécnica de Valencia, in 2010. Currently, he works as a research assistant within the Institute of Telecommunications and Multimedia Applications (iTEAM). His research interests include adaptive filtering and its applications to the active noise control.