The Impact of the Multi-core Revolution on Signal Processing

Alberto González[1], José A.Belloch[1], Francisco J. Martínez[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

This paper analyzes the influence of new multi-core and many-core architectures on Signal Processing. The article covers both the architectural design and the programming models of current general-purpose multi-core processors and graphics processors (GPU), with the goal of identifying their possibilities and impact on signal processing applications.

Keywords: Signal Processing, Multi-core processors, GPU, High Performance Computing, Parallel programming

1. Introduction

The current conception of Signal Processing is intimately linked with the type of computation required to perform the "Processing". In a recent issue of the IEEE Signal Processing Magazine [1], José F.M. Moura, president of the Signal Processing Society, noted: "As for processing, it comprises operations of representing, filtering, coding, transmitting, estimating, detecting, inferring, discovering, recognizing, synthesizing, recording, or reproducing signals by digital or analog devices, techniques, or algorithms, in the form of software, hardware, or firmware".

This definition emphasizes the strong dependence between signal processing and the computational media (digital or analogical, algorithms, hardware devices, software, etc) used to conduct it. In particular, if we focus on Digital Signal Processing, processors (in a wide sense) represent the most widespread digital devices in applications within this field.
The increase of processors performance and other digital devices has opened the possibility of addressing increasingly complex problems in a short period of time. This has been exploited both in real-time applications that are common in Signal Processing as well as other Signal Processing applications that require the management of very large data sets and which cannot be tackled within a reasonable time without the help of advanced computational tools. In summary, the advances of the hardware architecture of digital devices, including digital processors, strongly influence the techniques used and results produced in the field of Signal Processing. Considering the computational media, the following systems can be identified as the most used in Signal Processing during the past years:

General-purpose microprocessors (as those present in desktop computers, servers or high performance computers): The versatility, availability and ease of programming of these architectures have made them particularly useful in the field of Signal Processing, especially in intensive off-line applications.

Digital Signal Processors (DSP): They yield high performance as specific hardware for computationally intensive applications. They are especially appealing as components for the embedded market: devices that require intensive computing with small size, light weight, low cost and low consumption chips (GPS, mobile phones, etc.) [2].

Field Programmable Gate Arrays (FPGA): These are especially useful for real-time applications that require low weight, inexpensive, specific chips, with limited clock frequency, for highly repetitive operations (FPGA are used, for example, in space vehicles to cope with cosmic radiation), although it is difficult to use FPGAs as a general-purpose tool in a large variety of Signal Processing problems.

In the last five years, explicitly parallel systems are being accepted in all segments of the industry, including Signal Processing, in the form of multi-core processors and many-core hardware accelerators. The triple hurdles of power dissipation and consumption of air-cooled chips, little instruction-level parallelism (ILP) left to be exploited, and unchanged memory latency, combined with the desire to transform the increasing number of transistors dictated by Moore's Law into faster computers, has led the major hardware manufacturers to design multi-core processors as the primary means of increasing the performance of their products. General-purpose four-core chips from Intel and AMD are nowadays common in desktop machines, there exist six- and eight-core designs from these same vendors for the server market, and the ITRS Roadmap [3] predicts that by 2022 the number of general-purpose cores per chip will increase 100x with respect to current designs.

On the other hand, specialized (many-core) hardware with hundreds of simple cores is already available in the form of cheap, widely-spread NVIDIA and AMD/ATI graphics processors (GPU) incorporated in any standard graphics card. For example, 240 cores are embedded in NVIDIA GeForce GTX280 and, in the first quarter of 2010, the number of cores is promised to double in the upcoming NVIDIA Fermi design.

General-purpose multi-core processors (which we will refer here after as just multi-core processors) and specialized many-core accelerators like the GPU will surely impact current and future signal processing applications. First, these new hardware designs deliver an enormous high-performance computing capability at a remarkable low price, and programmers of signal processing applications will naturally want to exploit this. Second, as predicated by Herb Sutter in 2005, "The free lunch is over" [4]: Till 2004-2005 most classes of applications enjoyed free and regular performance gains, because the hardware manufacturers and computer architects reliably designed and produced ever-faster CPU. That enjoyable period is over and, although new processors yield higher performance, only those application developers who embrace parallel programming will benefit from it. In particular, Signal Processing is surely one of these applications that will be affected by the multi-core revolution.

The current conception of Signal Processing is intimately linked with the type of computation required to perform the "Processing".

As important as the hardware revolution may seem, it is the software that will determine the success or failure of the new products. A recent example is the IBM/Sony/Toshiba Cell B.E. processor, an innovative heterogeneous multi-core solution that did not succeed in the HPC arena mainly due to the lack of an appropriate, easy-to-use SDK-software development kit. Thus, the major hardware multi-core and many-core manufacturers dedicate considerable part of their efforts to develop and help others to create a varied ecosystem of low-level and high-level programming tools, which ease the task of software developers and, in the end, allow their designs to reach a larger number of customers.

INCO2 (www.inco2.upv.es) is a group created with the specific goal of tackling the software challenge in Signal Processing applications. INCO2 has been recognized as a research group in the Comunidad Valenciana (Spain) by the local government (PROMETEO 2009/013 project award). The research lines of INCO2 address problems of Signal Processing from an interdisciplinary perspective, providing solutions based on high performance hardware and developing algorithm design techniques that imply a modern and advanced software conception. Researchers of INCO2 have a vast experience in using parallel computing as a means of accelerating the time-to-solution and focus mainly on computers with multi-core and many-core architectures. The researchers of INCO2 are also part of the Partnership Program of NVIDIA Company, the world's leading manufacturer of graphics processing units.

The rest of the paper is organized as follows. The following two sections offer a brief description of the architectural characteristics of multi-core processors and GPU. Next, in Section 4, we discuss the possibilities of applying these architectures to the solution of Signal Processing problems, and we state logical needs and appropriate strategies needed to effectively tackle the problems. The final section of the article gathers our conclusions.

Probably the best form of appreciating the impact of the new architectures on signal processing is to analyze "possible application" and the performance reached in their solution when multi-core/GPU architectures are used. As a

The increase of processors performance and other digital devices has opened the possibil-ity of addressing increasingly complex problems in real time.

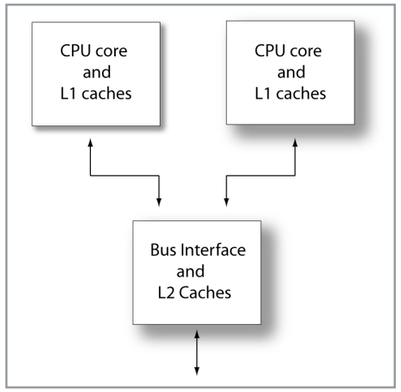

Figure 1. *Diagram of a generic dual-core multiprocessor*

continuation of this work, in the paper "Applications of Multi-core/GPU architectures in signal processing: some case studies" [5] we describe several case studies that show how parallelization on multi-core/many-core architectures can be applied to specific problems

2. The multi-core approach to parallelism

A **multi-core processor** or chip multi-processor (CMP) is an integrated circuit composed of two (dual core), four (quad-core) or more independent cores. Each core is an individual processor, but the cores in a chip may share certain resources as, e.g., a given level of the cache memory; see Figure 1.

A brief motivation of the multi-core development

Since the 1980s, microprocessors have dominated all computer markets, from embedded systems to servers, desktop computers and high-performance systems. Till 2004, the increasing number of transistors dictated by Moore's Law was exploited by system developers and computer architects to (respectively) reduce the scale of the chips (therefore, increasing their clock frequency) and produce more elaborated designs (e.g., with larger caches layered in multiple levels, more functional units, and, in general, capable of dynamically exploiting, i.e., at run-time, a higher amount of the instruction-level parallelism exiting in the codes).

In 2004, Intel joined all the other major hardware vendors (AMD, IBM and Sun) and declared the multi-core design as the main track to transform the gains dictated by More's Law into higher performance. The major reason for this is the limitation of the current semiconductor technology in terms of power consumption/dissipation, also known as the Power Wall. The acceleration of the clock frequency was a constant during this period: a VAX 8700 operated at 12.5 MHz while, 20 years later, an Intel Xeon reached 3.6 GHz (a factor of 290x). However, given the quadratic/cubic dependence between frequency and power dissipation of current CMOS technology, this trend came to an end: A chip operating at 5 GHz would simply melt!

Moving into the multi-core arena is not free as parallel programming must be explicitly addressed; however, this is currently recognized as the only way of pushing the performance of computer hardware, due to the combined effects of the power wall, the increasing gap between the processor and the memory speeds (the memory wall), and difficulties of finding enough parallelism in a single instructions stream to keep a single processor busy (ILP wall). Consider, e.g., that an increase of the clock frequency by 15% translates into a 2x power consumption but a potential increase in performance of only 15%. Whether this potential gain is real also depends on the ability of the programmer to hide the memory latency and the availability of more ILP in the program. On the other hand, by decreasing slightly the clock frequency, it is possible to double the number of cores in a design, maintaining the overall power consumption, and potentially doubling the potential performance gain. In this case, the potential gain is resulting from doubling the number of cores in a design is not hampered by the memory/ILP walls.

The multi-core solution is 10+ years old in the embedded market. Specific designs for mobile phones and network chips have included multiple cores for many years now. The big change is in the adoption of multi-core designs for the general-purpose market as well. Current multi-core chips for the server market include six-core AMD Opteron (model 2435, 45 nm scale, 75 W, 2.6 GHz, 128 KB L1 cache, 512 KB L2 cache, 6144 KB L3 cache), six-core Intel Xeon (model X7460, 45 nm, 130 W, 2.66 GHz, 9 MB L2 cache, 16 MB L3 cache), 8-core Sun UltraSPARC T1 "Niagara" (0.09 micron, 72 W, 1.2 GHz, 16I+8D KB L1 cache, 3 MB L2 cache), and AMD and Intel have announced, respectively, 12-core and 8-core designs for the first quarter of 2010. The number of cores is expected to double with each reduction in the integration scale (roughly, every two years), as long as Moore's Law holds.

CPU architecture

Current general-purpose multi-core processors feature basically the ISA (Instruction Set Architecture) of the corresponding uni-processor designs, with minor additions of synchronization instructions. The major consequence and advantage of sharing the ISA is the availability of legacy codes and a vast amount of programming tools for traditional uni-processor chips.

To increase performance, the processor datapath of current general-purpose processors is pipelined, so that the execution of multiple instructions can be overlapped. By splitting the processing of an instruction into a series of independent stages, with storage at the end of each step, instructions can be issued (to execution) at the processing rate of the slowest step, which is much faster than the time needed to perform all steps at once. Thus, pipelining improves the throughput of the datapath, but it does not decrease the execution time of a single instruction. The classic, simple pipelined datapath consists of four steps:

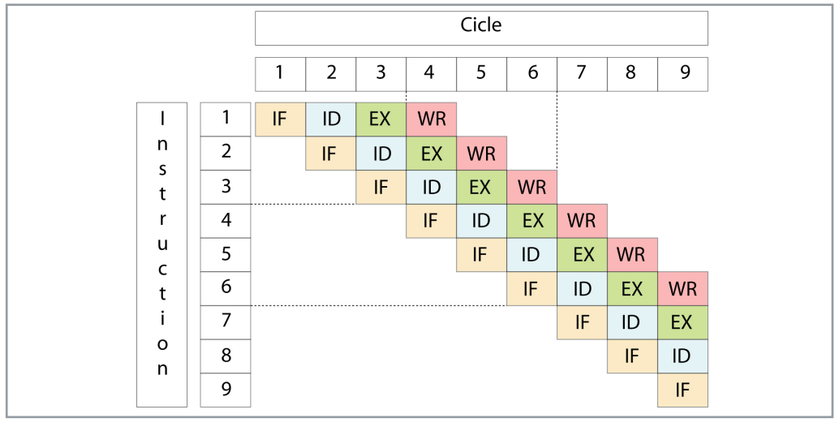

Figure 2. *Operation of a basic four-stage pipeline.*

1. Fetch instruction from memory (IF).
2. Decode instruction while, simultaneously, fetch the operands from the registers (ID).
3. Execute the operation (EX).
4. Write the result back in a register (WB).

The operation of such pipelined processor is illustrated in Figure 2. Intel stressed the concept of pipelining with 31 stages in the Prescott microarchitecture (February 2004).

The peak instruction issue rate yield by pipelined processors is 1 (instruction per cycle). To improve this performance, current processors issue more than one instruction per cycle; see Figure 3. Superscalar processors, like the Intel Xeon and the AMD Opteron multi-core designs, are the most spread class of multiple-issue processors. (VLIW processors, like the Intel Itanium2 are also multiple-issue architectures, but they issue a fixed number of operations encoded within one large instruction which explicit the parallelism among operations). Most general-purpose processors today are four and six-issue designs.

Superscalar processors detect and exploit ILP at run-time (dynamic scheduling), reordering the flow of instructions (out-of-order) to overcome the stalls due data hazards (i.e., data dependencies in the instruction flow). To be effective, this needs to be combined with a hardware-based speculation mechanism, which hides the stalls due to control hazards (due to branches in the instruction flow). The result is a complex hardware design, which requires substantial die area, and often is not power efficient. Because out-of-order multiple-issue processors are large and power hungry, few of them can be combined in a single chip. Thus, the current trend in multi-core design is to use simpler cores, with limited issue (e.g., 2-issue), with in-order scheduling, and moderate clock frequency.

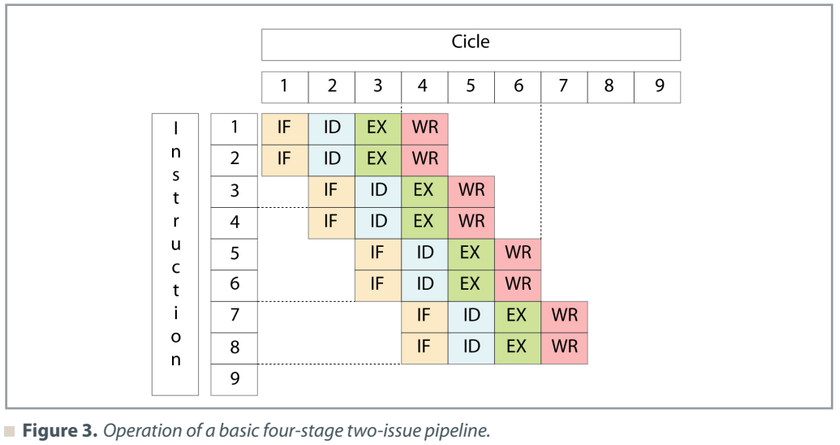

Figure 3. *Operation of a basic four-stage two-issue pipeline.*

The computational power of multi-core processors and GPU outweighs by a large factor that of the computers from past generations.

Memory system
The memory system plays an important role in multi-core processors, as the problem of feeding the processing units in the cores (memory bandwidth) is multiplied by the number of cores with respect to that of a uni-processor design. In general-purpose designs, caches are often made as big as the die area and power budget allow. As the number of transistors inside the chip increased, the number of levels in on-chip caches has increased with current processors from Intel and AMD featuring now a third level of on-chip cache. The first level of cache is usually (divided into data and instruction caches) small, fast and private to each core. Subsequent levels are (shared for data and instructions,) larger, lower, and in general shared by the cores.

Interconnect
Multi-core processors include a fast intrachip interconnect that provides the required communication path among cores and is responsible for maintaining cache coherence (if present). Simple, bus-based interconnect designs exhibit serious limitations in both bandwidth and latency and, therefore, cannot scale with the number of cores. Alternative network-on-chip (NoCs) designs, like the crossbar, overcome these limitations at the cost of a more complex design.

Cache coherence maintains a single image of the memory system (including the different caches and the main memory) and is a key issue as it determines the programming model that is natively supported. Broadcast-based coherence is simple and provides a solution, e.g., for up to eight cores in the Intel Core i7. Directory-based coherence allows multiple coherence messages to proceed concurrently and thus scales to a larger number of cores. In summary, we can provide the following list of advantages and drawbacks of the multi-core approach.

Advantages
- Existence of a large scopus of programming environments, libraries, tools and applications.
- Compatible with x86 ISA codes.
- Truly general-purpose.
- A restricted programming model.
- Moderate power consumption.

Drawbacks
- Suboptimal for many applications, specially data-parallel ones.
- High cost of large clusters (high price-performance ratio).
- High power-performance ratio.

3. The GPU approach to parallelism

A bit of history
Two interesting phenomena happened in the early twenty-first century: the video game market was positioned among the most vibrant ones and graphic processors were delivering an important computational performance. Graphic processors are very specific hardware in design and functionality. They yield high performance in applications for which they are designed, but the initial programming techniques in this class of processors were closely tied to the hardware. However, although graphic processors were and are hardware devices specially designed to carry out video rendering (vertex shader, primitive assembly, rasterizer, pixel shader, etc.), many of their features can be extrapolated with high efficiency to other applications.

When CUDA (Compute Unified Device Architecture) appeared in 2006, the development of GPU software changed significatively, becoming more accesible to non-specialized developers. In 2007, the functional units of the GPU turned into more general-purpose units. In the next two years, a large number of applications were addressed using GPU in a wide variety of fields [6]. Nowadays, we are attending to the generalized spread of GPU hardware, including multiprocessor systems built from GPU, the evolution of CUDA towards the OpenCL standard, etc. Nowadays general-purpose GPU (GPGPU) has become a powerful tool to the service of science and technology community.

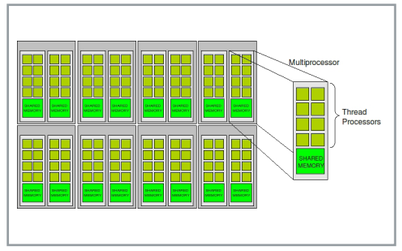

Figure 4. *Many-core architecture.*

Structure, Functionality and Programmability of GPUs
We can now view a GPU as a number of multiprocessors embedded in a chip. Each multiprocessor is made up of several fine-grain processors (or functional units). Each of these simple processors plays the role of a core in the current multi-core architectures. Although the clock frequency of the system is relatively low, the number of cores can be rather high, for example, 240 in the NVIDIA GT280. All multiprocessor cores run simultaneously a set of threads called warp and all of them execute, in principle, the same instruction (SIMT: Single Instruction, Multiple Thread), but each one on its own data (SIMD model: Single Instruction Multiple Data), as shown in Figure 4.

There are several classes of memory that can be accessed by the processors of the GPU: shared memory (accessible by all cores within a multiprocessor), global memory (read/write memory accessible by any core in any multiprocessor with a relatively high access cost) and constant and texture memory (read-only memory, closely related to the graphics processing). Communication between processors can be carried out through various types of memory, depending on the context.

The GPU is designed to operate in association with a CPU that plays the role of the "master processor (Figure 5)". The GPU is often connected with the master processor via the PCI-Express bus and all the communications between the GPU and the "outside" world happens through this bus. Thus, CPU and GPU form a dual system, where the GPU acts as a coprocessor or hardware accelerator.
Programming of GPU as general purpose machines is relatively complex, as it is partly tied to the low level aspects of the system (assembly language/hardware). However, the high performance delivered by these machines partially compensates for the difficult programming.

Follo[wing Flynn's classification [7], a GPU can be considered, from a conceptual point of view, as an SIMD machine (Single Instruction, Multiple Data); that is, a computer in which a single set of instructions is executed on different data sets. Implementations of this model usually work synchronously, with a common clock signal. An instruction unit sends the same instruction to all the processing elements, which then execute simultaneously this instruction on their own data, contained in a shared or local memory. This model differs from SPMD (Single Program, Multiple Data), which involves the simultaneous execution of the same program by several processors but not the same instruction. A SPMD program can have conditional statements (if... then...else) producing the execution of different operations on different processors depending on the index of the processor. This is not the case of SIMD machines.

The GPU programmer is in charge of generating the instructions to be executed in the GPU, sending them from the CPU along with data and, finally collecting the results. This requires a suitable programming environment that allows to easily implement such actions.

CUDA: an approach to a CPU-GPU architecture
Since 2006, GPUs are mostly programmed using CUDA (Compute Unified Device Architecture). According to NVIDIA (visit [6]): "CUDA™ is a general-purpose parallel computing architecture that leverages the parallel compute engine in NVIDIA graphics processing units (GPU) to solve many complex computational problems in a fraction of the time required on a CPU. It includes the CUDA Instruction Set Architecture (ISA) and the parallel compute engine in the GPU. In order to program to the CUDA architecture, developers can, today, use C, one of the most widely used high-level

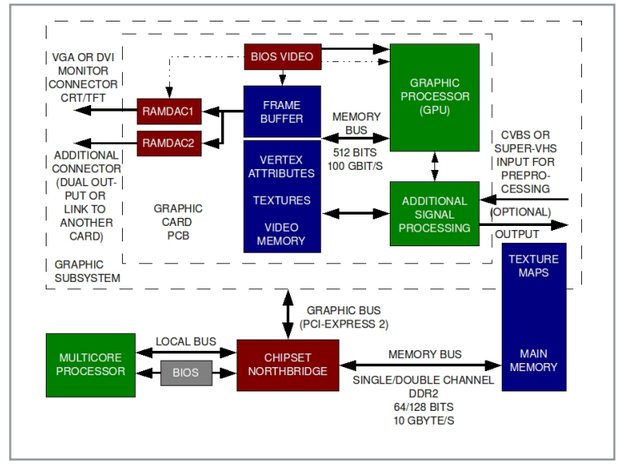

Figure 5. *GPU and CPU subsystems.*

programming languages, which can then be run at great performance on a CUDA enabled processor. Other languages will be supported in the future, including FORTRAN and C++".

CUDA provides instructions to transfer data and programs from the CPU to the GPU and to retrieve data back from the GPU to the CPU. It also provides a set of instructions for generating kernels (programs that run on the GPU only) which are arranged in the form of threads that are mapped onto the GPU cores.

CUDA has greatly simplified the job of programmers; however, its current development is not comparable to that achieved by standard compilers for other high-level languages/general-purpose architectures. The development and use of higher-level tools is strongly recommended. There exist several libraries that can address specific problems without having to write CUDA cores. This offers the programmer a high level programming style, similar to that commonly used in C or FORTRAN, hiding the tasks related with the implementation of GPU kernels inside library functions. While the degree of optimization has not yet reached that of standard libraries for general-purpose parallel computers, these preliminary tools represent an important aid in a not-too-friendly programming environment.

We can mention, for example, the following libraries: CUBLAS (implementation of the BLAS, Basic Linear Algebra Subprograms [www.netlib.org]), CUFFT (FFT package [6]), CULA [8] (implementation of the LAPACK [www.netlib.org] library), JACKET [9] (varied functionality of MATLAB), etc. There are also Integrated Development Environments that try to alleviate the programmer's task. One of the most significant is Parallel Nsight [6], developed by NVIDIA for the MS Windows programming environments (Visual Studio 2008). It allows debugging, profiling and analyzing GPU code using standard workflow and

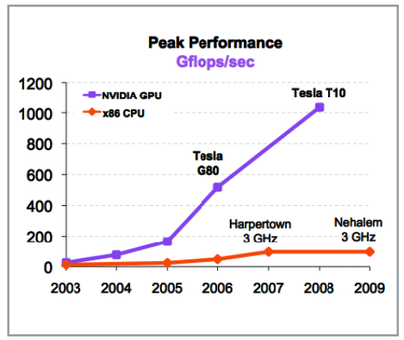

Figure 6. *GPU vs CPU GFlops.*

tools. Parallel Nsight supports CUDA C, OpenCL, Direct Compute, Direct3D, and OpenGL.

Performance

The performance of GPUs can be spectacular, especially if one only considers the peak performance of these machines. A proper use of the cores allows full concurrency, thus maximizing the whole power of parallelism (for example, 240 cores in the case of the GTX 280 card). This can potentially reduce execution times by an order of magnitude when compared with those achieved on a CPU; see Figure 6 obtained from [6].

However, several remarks are due here. Performance is much higher when using single-precision arithmetic. For example, on 2009 NVIDIA GPU processors, there is a single double-precision unit per multiprocessor; thus, e.g., only 30 double-precision units are present in a GT280. Furthermore, in a general application, the GPU attains a real performance that is typically much lower than its peak performance. To conclude this review of GPU, the following advantages and drawbacks can be remarked:

Advantages:

- Very high benefits in terms of Gigaflops/second.
- Excellent Price/Performance ratio.
- Existence of programming environments (CUDA, OpenCL...)
- Existence of libraries and tools.
- Many possible applications (see [5]).

Drawbacks

- A restricted programming model (SIMD model).
- CPU-GPU and I/O communications.
- Low-level programming.
- Insufficient tools.
- High power consumption.

4. Multi-core/many-core architectures in Signal Processing

Possibilities

From the discussion of multi-core processors and GPU in the two previous sections, it should be clear that the computational power of multi-core processors and GPU outweights by a large factor that of the computers from past generations. The new architectures also exhibit a more favourable power/price ratio, which may greatly facilitate their adoption and use in many application, even in those where the price may be a critical point. A preliminary conclusion is that the immediate future of computing, also in Signal Processing applications, seems tied to these architectures.

The Signal Processing field cannot remain indifferent to the computational advantages offered by the multi-core/many-core architectures. Indeed, these new systems can be an appealing alternative to the more traditional approach based on DSPs and FPGAs, as some practical Signal Processing applications have already shown; see, for example, [6]. Nevertheless, it must be yet established whether these architectures/tools are going to be widely incorporated as the primary choice in Signal Processing.

The adoption of a technology in a field of science or engineering may be influenced by factors other than the mere computational power provided by the hardware. For instance, programming models can strongly influence the pace and success of adoption. Also the nature and/or the scope of the problems may represent a limiting factor. As an example, some applications do not fit in the SIMD model, so that the use of GPU may not be appropriate or even viable; in some of these cases, the more flexible multi-core approach can solve the problem. Finally, the existence of a large scopus of legacy software or the lack of efficient software tuned for the new architectures can be a conditioning factor as well.

Only after a detailed analysis of these factors, it is possible to determine the usefulness of the multi-core/ many-core architectures in Signal Processing. Let us thus review the most popular programming tools and models available nowadays for the multi-core processors and GPU.

Multi-core is about running two or more actual CPUs (cores) on one chip. While these designs are not fundamentally different from previous multiprocessor architectures, the fundamental turning point lies in software development for applications targeting general-purpose desktop computers and low-end servers. In particular, the greatest software revolution in the past was the move from structured programming to object-oriented programming. The current "concurrency" revolution is an equally fundamental and far-reaching change in software development: Applications will only benefit from the continued exponential throughput advances in new processors if they are rewritten in terms of efficient concurrent (usually multithreaded) codes.

Luckily, there are many tools to help us in adapting software to the new architectures, especially with regular codes that are intensive in floating-point arithmetic like those frequently arising in signal processing applications. These tools can be classified in three major groups: compilers, languages/environments, and libraries.

Current compiler technology can expose a large fraction of the ILP providing a highly efficient base code for a single core. However, when dealing with multiple cores, compilers still need to be combined with some other tool (a language or an environment) that allows the programmer to pass additional information to the compiler. One such clear example is OpenMP [www.openmp.org], the current standard for shared-memory parallel programming valid for multi-core processors. OpenMP combines three elements: a high-level application programming interface (API), a compiler which transforms a program annotated with OpenMP directives into a multithreaded code, and a runtime environment combined with a library to assist in the parallel execution of the code.

OpenMP appeared in 1997 in response to the lack of a standard for parallel programming in shared-memory architectures that played a similar role to that of MPI for distributed-memory (message-passing) architectures. Version 3.0, released mid of 2008, includes the concept of tasks and the task construct, specially designed for multi-core processors. The new architectures have also given rise to a large number of contenders to OpenMP: UPC, TBB, Cilk, Chapel, etc. It still remains tobe seen whether any of these alternative solutions could become a real challenger to the acceptance of OpenMP as standard approach to programm multi-core processors.

CUDA is both NVIDIA's GPU architecture and the corresponding programming environment. Programmers use "C for CUDA" (C with NVIDIA extensions), compiled through NVIDIA C compiler, to code algorithms for execution on the GPU. CUDA architecture supports a range of computational interfaces including the new standard OpenCL [10]. High performance libraries for numerical computations, on the other hand, are much more mature. This is no surprising, as dense linear algebra kernels and the FFT have been traditionally employed by hardware vendors as the primary demonstrators of the performance attained by their designs. Current libraries for dense linear algebra include tuned multi-threaded implementations of BLAS by most hardware manufacturers (Intel, AMD, IBM, Sun, etc.), and higher level libraries as LAPACK and libflame. It is interesting to note that both LAPACK and libflame routines initially relied in BLAS to extract parallelism. However, the increase in the number of cores did require a redesign of these libraries, to extract a higher degree of (data) parallelism. The FFT has also received special attention over the last decades and, specially, with the multi-core revolution. FFTW, Spiral DFT and Intel MKL all include tuned implementations of the FFT. NVIDIA also provides its own libraries for dense linear algebra and FFT: CUBLAS and CUFFT. However, these are still suboptimal implementations which need to be further refined.

Applications

High Performance Computing (HPC) is broadening its scope to tackle a large variety of problems arising in many scientific and engineering areas. In Signal Processing, e.g., HPC techniques are applied to develop user applications in the promising market of processing, transmission and reproduction of multimedia content. The incorporation into the market of processors with multiple cores and the increasing use of graphics processors (GPU) in general-purpose applications, is at the same time a challenge and a great opportunity: the computing power of the new architectures may enable the solution of complex problems which require intensive computing using desktop computers, provided appropriate high performance algorithms are developed. The result is the availability of applications for the non-expert user that until recently were unthinkable in the consumer market.

Nowadays, signal processing has become a basic tool in many applications such as (re) creation and transmission of virtual environments, multichannel audio applications (recording and reproduction), wireless mobile communications systems with multiple antennas, to name just a few related with applications traditionally developed within the INCO2 research group. These applications often give rise to problems of high computational cost, even when using common signal processing algorithms, mainly due to the application of these algorithms to multiple signals and with real-time requirements.

The implementation of advanced algorithms for multichannel signal processing on new platforms based on computation-intensive architectures such as GPU and multi-cores is a scientific and technological challenge, of growing interest but unresolved at present, which will incorporate tools and possibilities currently available only in research to the user applications. In this line, the implementation of user systems require tools for massive signal processing: fast multichannel convolution, adaptive multichannel processing, MIMO channel equalization, and so on; as well as its interaction with computer algebra tools traditionally used in signal processing algorithms such as: solution of optimization problems with/out constraints on structured matrices, matrix decompositions (QR, SVD, etc), FFT, etc.

Only five years ago GPU supported a limited fixed number of functions, mainly addressed to the implementation of 3D graphics. Since then, GPU have evolved (both in its hardware implementation as in its programming interface - CUDA) to a very powerful processor, capable to carry out general tasks. Many references in the literature illustrate the generalized adoption of GPU, GP-GPU,

The number of cores is expected to double with each reduction in the integration scale (roughly, every two years), as long as Moore's Law holds.

General-purpose multi-core processors and GPU represent the future for many scientific applications, including signal processing as well.

also in applications other than image processing. Concerning the use of GPU for applications in digital audio processing, the oldest references date back to 2004 [11]. However, only very recently (2007 and 2008), the use of GPU has been employed in this area. The reason for this should be attributed to the GPU programming tools, quite complex in the beginning and with the constraint of using graphic processing procedures and terms: rendering, textures, etc. A second factor against the general adoption of GPU was that, for some time, the computational power provided by a general-purpose uni-processor was enough to give support for real-time applications.

Currrent proposals for possible applications of audio and acoustics on GPGPU include (www.gpgpu.org):

- Mixing audio signals.
- Modelling the acoustics of rooms and the Head Related Transfer Function (HRTF) for virtual environments.
- Adding sound effects (www.monalisa-au.org).

Other potential applications of digital signal processing on GPU can be found in the last paragraphs of [12] and [13]. An abbreviated list includes:

- Classical processing algorithms: FFT, convolution algorithms for solving differential equations, pattern recognition, sequence alignment (general algorithms using hidden Markov models), tracking.
- Algorithms for matrix massive computation: QR decomposition, Cholesky, SVD, etc.
- Wireless Applications: Implementation of some blocks of the physical layer, very suitable for standards based on OFDM, where FFT should be calculated (WiFi, WiMAX).

All the signal processing strategies developed to deal with a single signal or a few of them can be addressed when tackling multiple signals, taking advantage of its inherent parallel nature and the characteristics of the new hardware and software tools. One example of this is multichannel acoustic signal processing. This field has experienced a large growth in the last years, due to the increase in the number of sound sources used in new commercial applications for sound reproduction, and in the growing needs to include innovative effects and capabilities to the listening experience [14][15]. Moreover, the increasing market of advanced multimedia contents for home users creates the necessity of new multichannel sound processing tools, capable of extracting all the features that can be included in these contents. The creation of these contents requires as well multichannel signal processing tools, for stage analysis, signal filtering, noise reduction, etc.

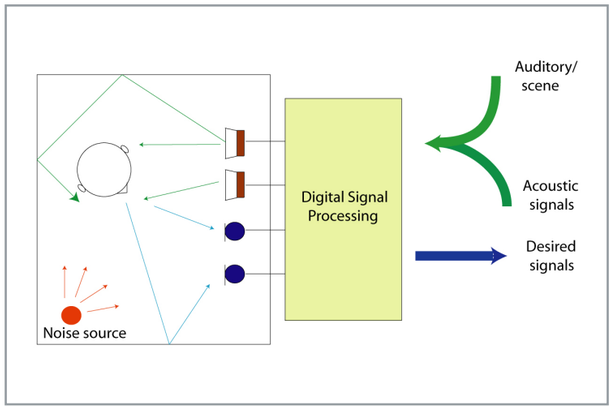

Figure 7. *Basic Scheme of a multichannel recording-reproduction system.*

The main multichannel recording-reproduction problem can be modelled as a discrete MIMO (Multiple Input, Multiple Output) system of sound signals. The problem of sound recording presents some analogies (in terms of multichannel signal processing) with the reproduction system; however, there are some distinct features. Particularly, it is possible to extract certain parameters from the analysis of the sound scene: number of sources, location of these, etc. A large number of applications of spatial sound can be derived from the scheme displayed in Figure 7, in applications that perform reproduction either through speakers or through headphones. Usually, we focus on reproduction through speakers (and recording by arrays of microphones) because this is the problem that poses harder scientific and technological challenges, and has a larger number of potential applications. In any case, it is always possible, from a generic viewpoint, to extrapolate the results obtained in reproduction by loudspeakers to headphones reproduction, if the physical phenomena and effects caused by the propagation of waves in the listening room are not taken into account.

From the acoustic man-machine interface depicted in Figure 7, which uses multiple channels for sound reproduction and acquisition, and in general can serve multiple mobile sources and listeners, the fundamental problems of signal processing can be identified. Some of these problems can be: multichannel acoustic echo cancellation, processing of signals from microphone arrays for beamforming, interference cancellation, signal separation, source localization, room equalization, active noise cancellation and spatial source location.

5. Conclusions

General-purpose multi-core processors and GPU will surely impact future signal processing applications, with the reason for this being twofold. First, these new hardware components exhibit a vast high-performance computing capability with a much favourable price-performance ratio, and programmers of signal processing applications will naturally want to exploit this. Second, only those application developers who embrace the explicit parallel programming model intrinsic to these architectures will benefit from the multi-core revolution.

It seems that GPU may not represent the optimal model for general-purpose parallel machines, but they state an important trend that other existing architectures (for example multi-cores) cannot ignore. In this sense GPU architectures are here to stay, either in its current form or as part of a hybrid design. As a special-purpose machines, GPU have assured their presence in the short and medium term. The market for video games and graphic applications is an appropriate field for GPU that provides the necessary economic support. Nonetheless, there are also important scientific and engineering problems which exhibit a considerable degree of data parallelism, which can benefit much from the important and cheap source of computational power in GPU. There is a considerable amount of literature that delves into this line of research, to which this paper is intended to contribute.

Multicore represents a good alternative for general-purpose parallel machine, with a simple and widespreath programming model and high performance in its application scope. However, it may not be the best approach for many real-time or fine-grain signal processing applications that require real time.

Both types of architectures represent the future for many scientific applications, including signal processing as well. In this paper we have reviewed the rationale and the state-of-the-art of both architectures. In a second part, we offer a brief description of some case studies developed within the INCO2 group that illustrate the use of the new architectures [5]. These examples aim at covering issues of low/medium level, with applicability in multiple Signal Processing applications.

Acknoledgments

This work has been 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] José M. F. Moura, "What is Signal Processing?," in IEEE Signal Processing magazine, vol. 26, no. 6, pp. 6-6, November 2009
[2] R. Schneiderman, "DSPs Are Helping to Make It Hard to Get Lost," in IEEE Signal Processing magazine, vol. 26, no. 6, pp. 9-13, November 2009
[3] International Technology Roadmap for Semiconductors, "International roadmap for semiconductors-System drivers," in http://www.itrs.net/Links/2009ITRS/2009Chapters_2009Tables/2009_SysDrivers.pdf, 2009
[4] Herb Sutter, "The free lunch is over: A fundamental turn toward concurrency in software," in Dr. Dobb's Journal, vol. 30, no. 3, March 2005
[5] A. Gonzalez, J. A. Belloch, G. Piñero, J. Lorente, M. Ferrer, S. Roger, C. Roig, F. J. Martínez, M. de Diego, P. Alonso, V. M. García, E. S. Quintana-Ortí, A. Remón and A. M.Vidal "Application of Multi-core and GPUs Architectures on Signal Processing: Case Studies," in Waves, vol. 2, 2010.
[6] http://www.nvidia.com/object/cuda_home_new.html
[7] M. J. Flynn, "Some computer organizations and their Effectivness," in IEEE Transactions on Computers, vol. 21 pp. 948–960, 1972
[8] http://www.culatools.com/
[9] http://www.accelereyes.com/
[10] Khronos Group, "OpenCL - The open standard for parallel programming of heterogeneous systems", in http://www.khronos.org/opencl , 2009
[11] E. Gallo and N. Tsingos, "Efficient 3D Audio Processing with the GPU", in ACM Workshop on General Purpose Computing on Graphics Processors, Los Angeles, August 2004.
[12] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone and J.C. Phillips, "GPU Computing", in Proc. of the IEEE, vol. 96, no. 5, pp. 879-899, May 2008.
[13] M.D. McCool, "Signal Processing and General-Purpose Computing and GPUs," in IEEE Signal Processing Magazine, vol. 24, no 3, pp. 109-114, May 2007.
[14] P.A. Nelson, F. Orduña-Bustamante, and H. Hamada, "Multichannel signal processing techniques in the reproduction of sound," in J. Audio Eng. Soc., vol.44, no 11, pp.973-989, November 1996.
[15] Y. Huang, J. Benesty, G.W. Elko, "Source localization, in Audio Signal Processing for Next-Generation Multimedia Communication Systems, Y. Huang, J. Benesty (Eds.), Kluwer Academic, Boston, MA, 2004 (Chapter 9).

Biographies

Antonio M. Vidal receives his M.S. degree in Physics from the Universidad de Valencia, Spain, in 1972, and his Ph.D. degree in Computer Science from the Universidad Politécnica de Valencia, Spain, in 1990. Since 1992 he has been in the Universidad Politécnica de Valencia, Spain, where he is currently a full professor in the Department of Computer Science. He is the coordinator of the project High Performance Com-

puting on Current Architectures for Problems of Multiple Signal Processing", currently developed by INCO2 Group and financed by the Generalitat Valenciana, in the frame of PROMETEO Program for research groups of excellence. His main areas of interest include parallel computing with applications in numerical linear algebra and signal processing.

Alberto Gonzalez was born in Valencia, Spain, in 1968. He received the Ingeniero de Telecomunicacion degree from the Universidad Politecnica de Catalonia, Spain in 1992, and Ph.D degree from de Universidad Politecnica de Valencia (UPV), Spain in 1997. His dissertation was on adaptive filtering for active control applications. From January 1995, he visited the Institute of Sound and Vibration Research, University of Southampton, UK, where he was involved in research on digital signal processing for active control. He is currently heading the Audio and Communications Signal Processing Research Group (www.gtac.upv.es) that belongs to the Institute of Telecommunications and Multimedia Applications (i-TEAM, www.iteam.es). Dr. Gonzalez serves as Professor in digital signal processing and communications at UPV where he heads the Communications Department (www.dcom.upv.es) since April 2004. He has published more than 70 papers in journals and conferences on signal processing and applied acoustics. His current research interests include fast adaptive filtering algorithms and multichannel signal processing for communications, 3D sound reproduction and MIMO wireless systems.

Francisco José Martínez Zaldívar was born in Paiporta, Spain, in 1966. He received the Licenciado en Informática and Ph.D. degrees from the Universidad Politécnica de Valencia, Spain, in 1990 and 2007 respectively. He is currently Lecturer at the Departamento de Comunicaciones, Universidad Politécnica de Valencia. His current research interests include parallel computing in signal processing.

Pedro Alonso was born in Valencia, Spain, in 1968. He received the engineer degree in computer science from the Universidad Politecnica de Valencia, Spain, in 1994 and the PhD degree from the same University in 2003. His dissertation was on the design of parallel algorithms for structured matrices with application in several fields of digital signal analysis. Since 1996 he has been a senior lecturer in the Department of Computer Science of the Universidad Politecnica de Valencia. He is a member of the High Performance Networking and Computing Research Group of the Universidad Politecnica de Valencia. His main areas of interest include parallel computing for the solution of structured matrices with application in digital signal processing.

Alfredo Remón was born in Valencia, Spain. In 2001 he received the B.S. in Computer Science from the Polytechnic University of Valencia, and the Ph.D. in 2008 from the Jaume I University of Castellón. He is currently an assistant researcher in the University Jaume I. His research interest include high performance computing of serial and parallel codes applied to dense linear algebra

Víctor M. García obtained a degree in Mathematics and Computer Science (Universidad Complutense, Madrid) in 1991, later an MSc degree in Industrial Mathematics (University of Strathclyde, Glasgow) in 1992 and a Ph. D. degree in Mathematics (Universidad Politécnica de Valencia) in 1998. He is a T.U. (senior lecturer) in the Universidad Politécnica de Valencia, and his areas of interest are Numerical Computing, parallel numerical methods and applications.

Enrique S. Quintana-Ortí is professor in Computer Architecture at the University Jaume I of Castellón, Spain.

His research interests are in parallel and high-performance computing and numerical linear algebra. Enrique has a PhD in Computer Science from the Polytechnic University of Valencia.

José Antonio Belloch was born in Requena, Spain, in 1983. He received the degree in Electrical Engineering from the Universidad Politécnica de Valencia, Spain, in 2007. In 2008, he worked for the Company Getemed (Teltow, Germany) as a software developer. His interest in parallel programming for Signal processing led him to enroll in a PhD program in 2009 with the Audio and Communications Signal Processing Group (GTAC). Currently, he is finishing a Master in Parallel and Distributed Computing at the UPV. He shares his studies in the Master with his research on multichannel Audio-Signal Processing onto the CUDA environment.