The impact of GPU/Multicore in Signal Processing: a quantitative approach

V.M García[1], A.Gonzalez[2] , C. González[2], F.J. Martínez-Zaldívar[2], C.Ramiro[1], S. Roger[2], A.M. Vidal[1]
Department of Information Systems and Computation[1] (DSIC),
Audio and Communications Signal Processing Group[2] (GTAC) iTEAM,
Universitat Politècnica de València - Camino de Vera s/n - 46022 Valencia (Spain)

Abstract

This paper presents a meaningful practical performance comparison between the last generation of Graphics Processing Units (GPUs) and the last generation multi-core CPUs when they are used to solve given Signal Processing algorithms. Two kinds of tests were considered: when GPU pre-designed computational libraries were available, and when the GPU code was developed by the authors. Results show that GPUs offer great possibilities, but its programming is still hard and high performances can be obtained only if the algorithm can be adapted to the GPU programming model.

Keywords: Multi-core/GPU Architecture, performance evaluation, MAGMA, QR, K-Best algorithm, MIMO decoder

1. Introduction

In last years, the number of scientific contributions and research projects related to the use of Graphics Processing Units (GPUs) as general purpose computers (GP-GPU) has significantly increased. This phenomenon has occurred in almost all engineering fields that require intensive computing, and Signal Processing is not an exception [1].

The initial idea in many of these applications and projects is that GPU (many-core) can achieve better performance than CPU (multi-core) due to a simple quantification of "many" against "multi", and assuming that CPU are more expensive and more resource-consuming. Perhaps this belief stems from the graphs distributed by NVIDIA (see Figure 1) [2], where GPU performance in terms of Gflop/s is quite impressive. However, it must be noticed that the performances of sequential or parallel computers are always obtained under specific conditions and the results should not be extrapolated to other circumstances.

GPUs are fascinating tools and represent a quantitative leap in the development of high performance hardware. Probably it would be impossible to go back, and the near future cannot be imagined without GPU technology. However, to give accurate opinions and ensure their usefulness, it is essential to carry out a quantitative performance analysis. It should show the ranges and working conditions for which the attained benefits are really important, together with a quantitative assessment of those benefits.

GPUs exhibit a fast evolution. The last models on the market have more cores, more computational power and several new features [3][4]. Software tools also allow a friendlier GPU programming than some years ago: CUDA (Compute Unified Device Architecture, [2]) is continuously evolving and the most recent SDK versions solve problems of previous versions; OpenCL [5] language seems a good future alternative but unfortunately it does not show the evolution speed and the performance of CUDA.

However, there remain serious programming difficulties, especially if GPUs are intended to be used as general purpose machines. Sometimes these difficulties are intrinsic and related to the SIMD (Single Instruction Multiple Data) model, which essentially allows data-parallelism but limits the use of graphics accelerators in general purpose applications. Instead, MIMD (Multiple Instruction Multiple Data) models allow task-parallelism and can be more efficient in this last type of applications.

Some GPU limitations are derived from their technical specifications (for instance, their clock frequency is lower than the current-generation CPU), from usage and memory capacity limitations and, especially, from the fact that GPUs exist as accelerators and not as chips that include all the features of the set CPU-GPU. Until now, the attempts to design this kind of chip (see, e.g., reference [6]) have resulted only in theoretical approaches and prototypes that could not get into commercial stage.

A realistic approach to the performance of GPU in a particular field involves identifying the problems that can be solved with these tools, defining which benchmarks will be used in performance analysis and setting evaluation metrics. In this paper we pursue to show a quantitative analysis of the benefits obtained by the GPU in given applications of Signal Processing. The work is a natural continuation of the previous papers [7][8], which analyzed the impact and the potential that these architectures may represent in the field of Signal Processing. In the present paper we analyze the performance of some particular libraries used in Signal Processing with a more pragmatic point of view.

We chose a simple series of typical problems found in several applications in this field and assessed their behavior when they are solved on current CPU and also on computers that incorporate GPUs. The last two GPU models marketed by NVIDIA, Tesla and Fermi [3], as well as the latest generation Intel processors with 4 and 6 cores [9], have been used for this purpose.

The chosen target problems are described in what follows. First, we chose two general problems with application in Signal Processing: the linear Least Squares problem (LLSP) and the QR Decomposition [10][11], which can be considered as the computational tool to solve LLSP. The other is an algorithm employed in the detection stage of MIMO (Multiple Input Multiple Output, [12]) wireless systems: the K-best tree-search detector [13], which presents high computational requirements and a different approach.

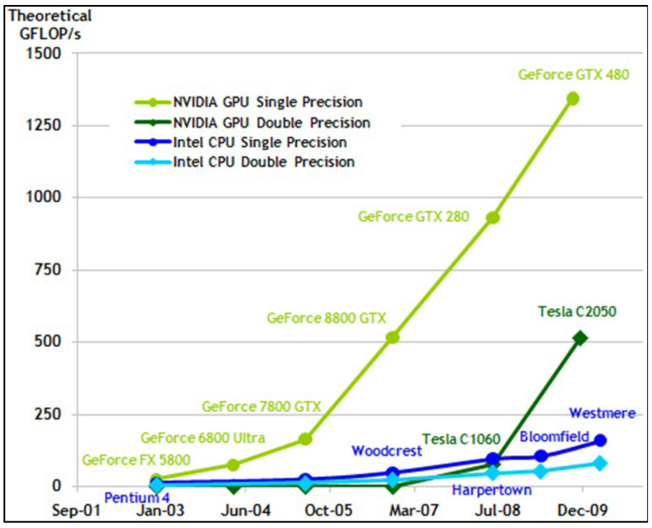


Figure 1. *Evolution of the theoretical floating-point operations per second achieved with different GPU versions [2].*

The chosen problems cover several aspects of usability of GPUs in Signal Processing. The first two problems are very general and there exist libraries that can solve efficiently them on both GPU and CPU. In this work we have employed the LAPACK library [14] for CPU and the MAGMA library [15] for GPU. The other problem is more specific, thus, the necessary code for their implementation has been developed by the authors.

The rest of the paper is organized as follows. Section 2 describes the evolution and current status of multi-core/ many-core architectures. Section 3 presents the computational problems studied here and describes in detail the used benchmark. The different computational experiments carried out and the conditions and results are shown in Section 4. Section 5 discusses these results. Finally, Section 6 reports the conclusions of present work.

2. GPU/multicore models evolution

2.1 Heterogeneous computational models

One of the more decisive concepts for successfully programming a computer that uses GPU is the underlying model of parallel computer. Traditionally, a GPU card has been considered as an isolated parallel computer, fitting a SIMD model, and connected to a sequential computer

GPUs represent a quantitative leap in the development of high performance hardware. However, there remain serious programming difficulties, especially if GPUs are intended to be used as general purpose machines.

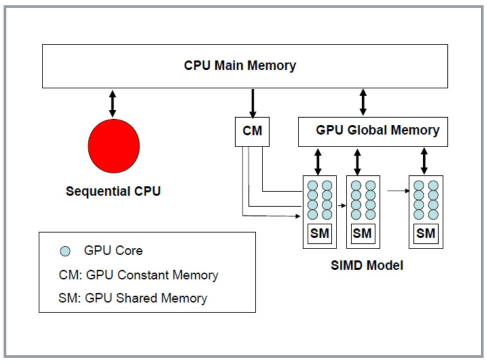


Figure 2. *Sequential computer with a GPU accelerator.*

(see Figure 2). From this point of view, the GPU card can be seen as a set of 240/480 (depending on model) processors, running the same instruction simultaneously, each one on its own set of data. An appropriate performance metric for this system could be the speedup achieved by the graphic card against the CPU. This speedup can be obtained by dividing the runtime of a program executed in the sequential processor by the execution time given by the graphic card. A similar metric consists of comparing the Gflop/s needed by the CPU and the GPU to solve the same problem. Note that the different clock speed of operation in the sequential unit and in the GPU causes an unfair comparison between performances of CPU and GPU. In fact, this is not a classical speedup because the considered sequential machine is not an instance of the considered parallel machine for just one processor.

A more realistic model should consider the host system and graphic card as a whole, and the host computer as another parallel computer, at the same level than the GPU. This leads us to the heterogeneous parallel computer model. A similar model is used for instance in [16]. Following this idea, a system with a GPU or an accelerator card (see Figure 3) consists of a set of two (or more) parallel computers, with different speeds, each of them with access to different types of memory, which also implies different memory access times for each processor.

A model of this kind would be characterized by the number and type of processors, and different access time of each processor to the different types of memory. For example a system comprising a multicore type CPU is considered in Figure 3. This system has a first-level cache and a main memory, shared by all cores, and an accelerator manycore type card with different types of memory (global memory, constant memory, shared memory). In this case the CPU can write and read to global and constant memory of the GPU and GPU can write and read to its global memory and only read from constant memory.

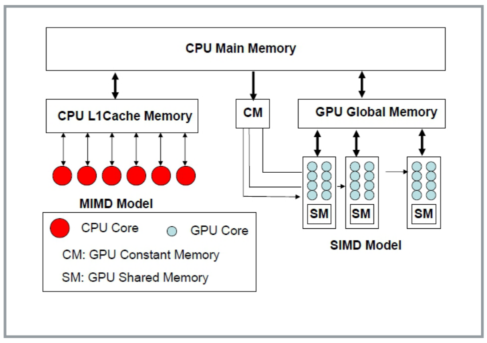


Figure 3. *Heterogeneous parallel computer system.*

Performance and programming of this model depend on: the type of parallel computer (MIMD in the case of the CPU, SIMD in the case of the GPU), the clock speed of CPU and GPU, the access time to each type of memory and the amount of memory in each memory class. Note that the performance of a GPU in a system of this kind is difficult to evaluate as an isolated component. The best metric in this case may be to compare the speed of the system with and without the accelerator card. It must be allowed (and even encouraged) a simultaneous use of the GPU and CPU, and compare the Gflop/s obtained when they act together and when eliminating the use of the GPU to solve a concrete problem.

This second approach is much more realistic, and it is used, for example, in the case of numerical linear algebra libraries like MAGMA or CULA [17]. Although these libraries are considered specific libraries for GPU, they run usually part of their programs on the CPU and reserve the GPU to execute those parts that exhibit a strong data parallelism.

The systems used in our case can be modelled as follows: two benchmarks are mapped onto a heterogeneous model system (QR and LLSP) and the other one uses the GPU model system (K-best problem). The specific technical characteristics of the computers are described later.

2.2 Last and penultimate GPU generations

Three GPU generations have been released since the beginning of NVIDIA GP-GPU computing until nowadays: firstly Tesla 8-series, then Tesla 10-series, and finally Tesla 20-series (or Fermi architecture). The most important differences between Tesla 10-series and Fermi [18] are summarized below.

2.2.1 Hardware features

- The number of multiprocessors of the GPU has decreased from 30 to 14, and the number of cores per multiprocessor has increased from 8 to 32 cores, thus the total number of cores has increased from 240 to 448.

- Memory error control (ECC) is supported in global and cache memory to improve reliability.

- Configurable size L1 cache exists (either 16 kB or 48 kB per core). Shared memory size is now either 48 kB or 16 kB depending on the chosen cache size. The L2 cache is 768 kB per multiprocessor (the same size for maximum total L1 and L2 caches).

• GDDR5 DRAM is used instead of GDDR3 DRAM.

2.2.2 Base architecture

• The number of threads per block has been doubled (from 512 to 1024), although the maximum number of blocks per multiprocessor remains constant (8). The maximum number of threads per multiprocessor has increased from 1024 to 1536 but the number of threads per warp remains constant (32).

• There were 32-bit/16k registers in 10-series and 20-series present 32-bit/32k registers per multiprocessor.

• Double precision arithmetic performance has been improved (now it is IEEE 754-2008) with half the speed of single precision arithmetic (in Tesla 10-series this relative speed was 1/8th).

• Virtual address space of 64 bits with unified address space (except for constant and texture memory) is available for 64-bit versions of Linux and Windows operating systems.

• Atomic instructions are now faster because atomic values can be placed in L2 cache.

• There exists a new instruction set transparent to the programmer thanks to the PTX code.

But not just hardware and architecture modifications improve the execution times of applications. CUDA 4 helps to get better performance:

• Grids can be tridimensional (in Fermi).

• Several GPUs can be shared across multiple threads: concurrent kernels can be launched from different host threads. A single thread can access all GPUs.

• Changes in page-locked host memory allocation reduce memcpy overhead and usage of system memory.

• Data can be copied from one GPU to another without intervention of CPU (only in Fermi).

• Support for unified virtual addressing: single memory space in the GPU+CPU system has been implemented on Tesla 20-series and with 64-bit applications on Linux and Windows.

• Improvements in performance analysis debugging and disassembling have been developed.

2.3 Last multicore generations

During the last few years, Intel and AMD have been focused in developing up to 12 cores processors [9][19][20]. The last Intel architecture, codename Nehalem, uses 45 nm technology (Westmere is the 32 nm evolution) in processors with 2, 4, 6 and 8 cores. On the other hand, AMD developed the Phenom architecture for desktop environments, with 45 nm technology and with 2, 3, 4, and 6 cores, and Opteron 6000 series for servers, with 8 and 12 cores.

In both cases, their purpose is getting maximum performance with minimal consumption using hyperthreading and other technologies as:
•Intel's Turbo Boost Technology and Intelligent Power Technology allow the processors to change their frequencies on demand in order to improve their performance (if more power is needed) or to save energy (in low utilization periods).

• AMD's Turbo Core Technology allows to switch off some cores in order to increase the frequency of the other ones when this provides a better performance.

However, new objectives of both companies have appeared due to the GPU growing market in recent years. Intel started the Larrabee architecture GPGPU project some years ago with an uncertain future. Nowadays its interest seems focused on the Teraflop Research Chip (also known as Polaris), which is an 80-core processor with a GPU-like conception [21].

On the other hand, after acquiring ATI and since 2006, AMD has been developing the 40 nm technology known as AMD Fusion, which integrates the processor and the GPU on the same chip [22].

One of the more decisive concepts for successfully programming a computer that uses GPU is the underlying model of parallel computer.

3. Benchmarks

3.1 Problem description

3.1.1 QR decomposition

Also known as ***QR*** factorization [10], it decomposes a matrix ***A*** into a product of an orthogonal matrix and an upper triangular matrix, ***A = QR***, where ***Q*** is the orthogonal matrix (unitary in case of ***A*** being complex) and ***R*** is an upper triangular matrix.

More generally, we can factor a *mxn* matrix, being $m \geq n$, as the product of the matrices ***Q*** and ***R***. Since the last *m-n* rows are full of zeroes, the following partition is commonly made:

$$\mathbf{A} = \mathbf{QR} = \mathbf{Q}\begin{bmatrix} \mathbf{R}_1 \\ 0 \end{bmatrix} = [\mathbf{Q}_1, \mathbf{Q}_2]\begin{bmatrix} \mathbf{R}_1 \\ 0 \end{bmatrix} = \mathbf{Q}_1\mathbf{R}_1, \tag{1}$$

where $\boldsymbol{R}_1$ is a *nxn* upper triangular matrix, *Q1* is mxn and $\boldsymbol{Q}_2$ is *mx(m-n)*. $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ both have orthonormal columns.

All the problems for the general use of GPU are being solved through: libraries like MAGMA and CUBLAS, new programming environments such as CUDA, and new hardware improvements such as the new Fermi card.

This problem can be solved in a GPU by using the MAGMA library, which implements three versions of the QR decomposition, all of them based on elementary reflectors. As commented above, they follow an hybrid model, which means they are not totally run on GPU. The tiniest and/or hardest tasks to be parallelized efficiently are executed on CPU using LAPACK/BLAS, while the rest of the code (like BLAS 3 operations) are calculated on GPU using CUBLAS [23]. This method provides more overlapping between CPU and GPU works and improves algorithm efficiency.

The functions related to each of the three versions, whose interfaces are equivalent to the LAPACK ones for CPU, are described below.

3.1.1.1 Xgeqrf

The matrix and the final result are stored in CPU memory and the routine allocates memory in the GPU and makes all the necessary transfers in both directions.

```
magma_int_t magma_Xgeqrf(magma_int_t m,
magma_int_t n, <type> *a,
            magma_int_t lda, <type> *tau,
            <type> *work, magma_int_t lwork,
magma_int_t *info)
```

3.1.1.2 Xgeqrf_gpu

In this version, matrix *dA* must be already stored in GPU memory, where it also stores the results and the triangular factor of the block reflector matrix used in the factorization so they can be used later avoiding their recomputation and transfers, therefore the application of Q is much faster.

```
magma_int_t magma_Xgeqrf_gpu(magma_int_t m,
magma_int_t n, <type> *dA,
            magma_int_t ldda,<type> *tau,
            <type> *dT, magma_int_t *info)
```

3.1.1.3 Xgeqrf2_gpu

This version is similar to Xgeqrf_gpu: matrix dA is allocated in GPU memory but the triangular factor of the block reflector is applied but not stored for future use.

```
magma_int_t magma_Xgeqrf2_gpu
(magma_int_t m, magma_int_t n, <type> *dA,
            magma_int_t ldda, <type> *tau,
            magma_int_t *info)
```

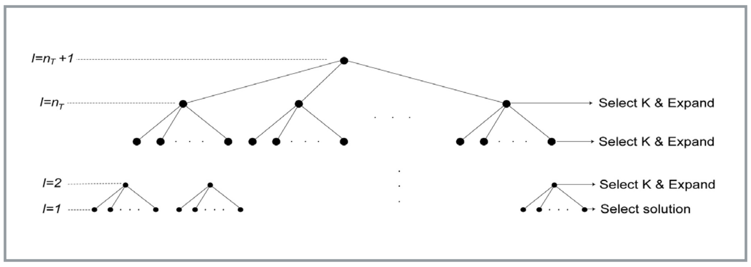


Figure 4. *Description of the search tree in a K-Best MIMO detector.*

3.1.2 Linear least squares problem (LLSP)

This is equivalent to solving an overdetermined system of equations. Given $\mathbf{A}\in\Re^{mxn},\mathbf{x}\in\Re^{n},\mathbf{b}\in\Re^{m}$, the LLSP consists of finding an $\mathbf{x}_{LS}\in\Re^{n}$ so that $\|\mathbf{A}\mathbf{x}_{LS}-\mathbf{b}\|_2 = \min\|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2$.

It is common to use the QR factorization in order to solve this (full-rank) problem [10].The MAGMA implementation is based in the *Xgeqrf_gpu* model (described in the section above).

The LS interfaces are the same as in the LAPACK library:

```
magma_int_t magma_Xgels_gpu(char trans,
magma_int_t m, magma_int_t n,
            magma_int_t nrhs, <type> *dA,
            magma_int_t ldda, <type> *dB,
            magma_int_t lddb, <type> *hwork,
magma_int_t lwork, magma_int_t *info)
```

3.1.3 K-Best tree-search detection in MIMO wireless systems

One of the tasks of the receiver in a multiple-input multiple-output (MIMO) wireless system is the detection of the transmitted data, which is affected by the communication channel and the noise [24]. Given the received signal **x** and the channel matrix **H**, the detection problem consists in determining the transmitted vector s with the highest a posteriori probability. In practice, this is carried out by solving the following integer least squares problem

$$\hat{\mathbf{s}} = \arg\min_{s\in M^{n_T}}\|\mathbf{x}-\mathbf{H}\mathbf{s}\|^2, \tag{2}$$

which can be straightforwardly solved by an exhaustive search over the whole set of n_T -dimensional lattice points s, a priori known and denoted by $M^n{}_T$. Note that n_T and n_R stand for the number of transmitting and receiving antennas, respectively.

This implementation is cumbersome for practical systems; however, its complexity can be substantially reduced by means of tree search detection methods. Assuming that $n_T = n_R$, the QR factorization (1) of the channel matrix $(\boldsymbol{H} = \boldsymbol{QR})$ transforms the system into an equivalent one that can be solved using a tree structure [25]. If (2) is multiplied by $\boldsymbol{Q}^T$ and $\boldsymbol{y} = \boldsymbol{Q}^T\boldsymbol{x}$, problem (2) can be equivalently expressed as

$$\hat{s} = \arg\min_{s\in M^{n_T}}\|\mathbf{y}-\mathbf{R}\mathbf{s}\|^2 = \arg\min_{s\in M^{n_T}}\sum_{l=1}^{n_T}\left|y_l-\sum_{j=l}^{n_T}R_{lj}s_j\right|^2, \tag{3}$$

where the triangular structure of $\boldsymbol{R}$ has also been exploited.

The detection process starts from the $l = n_T$ level of the tree and each survivor candidate of the *l-th* level is represented by $S^{(l)} = [s_l,s_l+1,\ldots,s_{n_T}]$

and called tree node. The accumulated partial Euclidean distance (PED) associated to $S^{(l)}$ is recursivelv calculated as $D_l\left(S^{(l)}\right)=D_{l+1}\left(S^{(l+1)}\right)+\left|e_l(S^{(l)})\right|^2$, where $\left|e_l(S^{(l)})\right|^2=\left\|y_l-\sum_{j=l}^{n_T} R_{lj}s_j\right\|^2$ is the distance between levels l and $||l + 1||$ in the decoding tree, which will be named branch weight, with $D_{n_T+1}\left(S^{(n_T+1)}\right)=0$. Hence, the solution of (3) is the vector $S^{(1)}$ that minimizes $D1(S_{(1)})$.

Every time we descend from a node in level l (parent node) to the nodes in level l-1 that are connected to it (children nodes), the branch weights of the children nodes are computed, then, it is said that the parent node has been expanded. Various strategies can be followed to discard parts of the tree where the candidates are likely to be far from the solution of (3). The K-Best algorithm [13] expands the detection tree from top to bottom and considers only those K survivor candidates that show the smallest accumulated PEDs at each level of the tree (see Fig. 4).

The main advantage of this method is that the maximum number of expanded nodes is limited by K and can be known in advance, which determines the necessary resources and makes its hardware implementation easier. Also, some parts of the algorithm as the node expansion of all the surviving candidates at the same level can be carried out in parallel.

3.2 Benchmark descriptions

3.2.1 QR decomposition and Linear Least Squares problem

The MAGMA distribution includes several source-code files which allow a performance analysis of each of the operations supported by the library, also including a comparison with a pure CPU-execution using LAPACK. Some of these tests evaluate the performance of the subroutines without considering initial/final data transfers between CPU and GPU, because the routines they use suppose that the data are already in the GPU. Other tests take into account this time because the used subroutines assume that the initial data is still in CPU memory. Variability of performance when different datatypes (real or complex) and precision (simple or double) are used has also been tested.

3.2.2 K-Best tree-search detection in MIMO wireless systems

In the K-Best problem, at every level of the detection tree, the number of expanded nodes is $N = K \times M$, where M is the constellation size. For each node, the branch weight must be computed, the accumulated distance updated and then the best *K* survivors selected to proceed in the next level.

In the CUDA implementation, we assigned the calculations of the PED in each branch of the tree to a different thread. When all threads finish the calculations, the information of the N nodes and their distances are sent to the CPU to obtain the K minimum distances. The inherent dependency between the levels of the tree requires barrier synchronization among threads. This fact also makes impossible to employ shared memory to store the calculations and to get the K survivors in the kernel, since the shared memory is only accessible by a certain processor and it would be impossible to operate with distance values residing on different processors.

Therefore the CUDA parallelization includes only the calculation of the cumulative distance for every node at a certain level and does not include the sorting and calculation of the K-best survivors, this process are carried out sequentially.

4. Experiments

4.1 Computer description

Two different computers have been used in our experiments. The first one, denoted as System 1, has two four-core processors: Intel Xeon E5430 @ 2.66 GHz with 6 MB cache memory. The installed graphics card is a Nvidia Quadro Fx 5800 with 30 8-core multiprocessors (240 cores), a clock frequency of 1.35 GHz and 4 GB of GDDR3 global memory. The GPU hardware allows concurrent copy and execution and the installed CUDA SDK and Toolkit is the 3.1 version. The machine has the MKL 10.1, CULA 2.1 and MAGMA 1.0 RC4 libraries installed and has support to work with OpenMP and UPC.

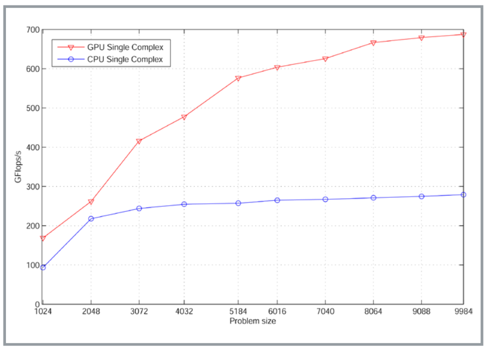
The other computer, denoted as System 2, has two Intel Xeon X5680 processors at 3.33 GHz and 96 GB of GDDR3 main memory. Each one is an hexacore processor with 12 MB of cache memory, and with the hyperthreading technology they have 24 virtual processor. It contains two Nvidia Tesla C2070 GPUs with 448 cores and 6 GB of GDDR5 global memory each one. The core frequency is 1.15 GHz. The architecture of these GPUs is Fermi and hence it supports the maximum parallelism level with several kernel execution overlapping, data copy and kernel execution overlapping, simultaneous host to device and device to host data copy, etc. The installed CUDA toolkit and SDK version is 4.0 and it has also libraries as MKL 10.3, CULA 2.1 and MAGMA 1.0 RC4 installed.

4.2 Performance Results

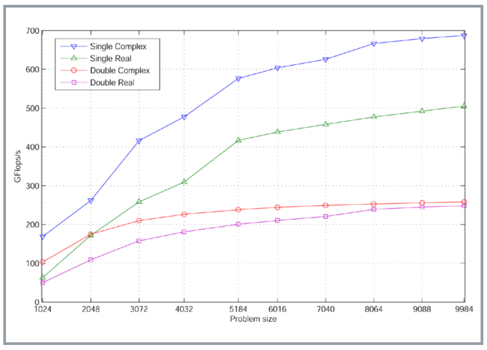
4.2.1 QR decomposition

This test has been run over **System 2.** The sizes (number of columns or rows of the used square matrices) were: 1024, 2048, 3072, 4032, 5184, 6016, 7040, 8064, 9088 and 9984. Matrices were randomly generated from a normal distribution.

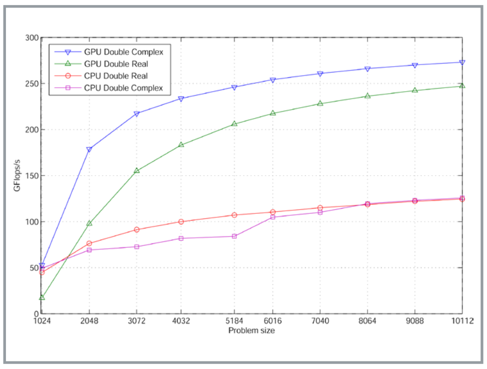
Since all MAGMA methods for QR factorization present similar results, it is just shown the *Xgeqrf2_gpu* evolution, which offers a subtle better performance when comparing with LAPACK for single precision complex data, even though for the smallest matrix sizes the difference is not large. When the matrix size grows the GPU works more than 400 Gflop/s faster than the CPU (see Fig. 5).



■ **Figure 5.** *Comparison with LAPACK QR using single precision complex*



■ **Figure 6.** *Xgeqrf2_gpu performance comparison depending on the datatype and including data transfers.*



■ **Figure 7.** *LS performance for double precision complex and real datatypes.*

Figure 6 shows performance differences among different datatypes. The fastest operations are in single precision, and complex datatypes offer better performance than real ones.

These results are similar to the ones shown in the MAGMA project website [26], showing only little differences due to the use of Tesla C2070 cards instead of the C2050 model and different CPUs, and also considering data transfers (while the official results measure only the calculations). See [27] for the results published by MAGMA developers.

4.2.2 Linear least squares problem

This test was run over System 2, using the following sizes of square matrices: 1024, 2048, 3072, 4032, 5184, 6016, 7040, 8064, 9088 y 10112. Matrices are randomly generated from a normal distribution.

Figure 7 shows performance comparison with LAPACK for double precision operations. For complex data, both the CPU and GPU start almost at the same measure but when size 2048 is reached the GPU increases performance a lot. Even though after that point their evolution is similar, they keep separated by 300 Gflop/s. For real data, CPU works better for small matrices but it stalls and is easily overwhelmed by GPU when size 2048 is reached.

Regarding the datatype, it is shown again that single precision datatype are processed much faster than double precision datatype, and that complex data gets better performance than real data (see Fig.8).

For LS problem, the MAGMA website just shows a comparison for complex datatypes, again with similar results to the ones shown. See [28] for the results reported by MAGMA.

4.2.3 K-Best tree-search detection in MIMO wireless systems

These benchmark tests were performed on the two systems described before. They were carried out by varying the different parameters of the problem, such as channel matrix size and the constellation size. The selected sizes for $\boldsymbol{H}$ were 2x2 and 4x4 and the constellation sizes M={4, 16, 64, 256}. Different values for K were considered depending on the constellation size. Furthermore, a multicarrier transmission with different number of subcarriers, denoted as N_c, was considered. The values for these two last parameters will be addressed in the Results section.

For the CUDA implementation, we defined a two-dimensional grid containing two-dimensional blocks of size N_H = 16x16, so the total number of blocks required can be calculated as follows:

$$N_B = \frac{N_c \times K \times M}{N_H}$$

and the number of blocks in each dimensions is upper bounded by $\sqrt{N_B}$.

Figure 9 shows the speedup resulting from the comparison between the computational times to run the algorithms at the GPU and the computational times of the implementations on CPU. The performance offered by Fermi is twice the performance with the Tesla architecture. Note that the code is not optimized to make use of new benefits and features of the Fermi architecture, but it is a code implemented for previous GPU families.

This implementation considers that only the PED calculation of the tree branches was paral-

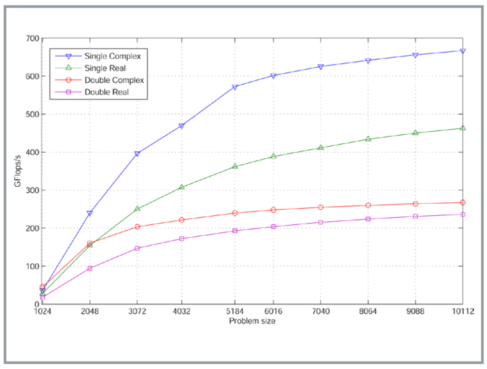


Figure 8. *LS performance comparison depending on the datatype and including data transfers.*

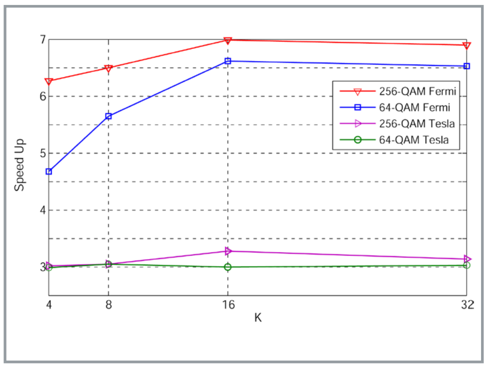


Figure 9. *Speedup for 400 subcarriers in Nvidia Tesla and Fermi architectures.*

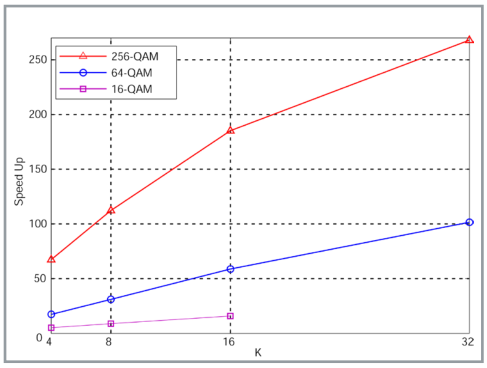


Figure 10. *Speedup for 1 subcarrier in Nvidia Fermi architecture.*

lelized, and we see that the tree branches are processed up to seven times faster on the GPU than on the CPU.

However, if we measure the data transfer time between the GPU and the CPU at each level of the tree, the total computational times of the algorithm are very similar in the sequential and in the parallel implementations.

Therefore, the benefits here are much lower than those achieved in the previously analyzed benchmarks. This is because the previous tests were carried out with highly optimized computing cores, and also the scheme of such problems allows a simultaneous use of the CPU and GPU when the operations are independent.

In the K-Best problem, the K-best survivors' calculation cannot start until all threads have finished the previous GPU computation and thus the CPU remains idle in the meantime. In the same way, the GPU cannot proceed with the following branches of the tree until the CPU has not determined the new K survivors and sent such information back to the GPU.

A useful measure of the goodness of a CUDA implementation is the degree of parallelization of the algorithm on the GPU, which is revealed by the speedup achieved with many GPU threads against a single thread. In order to achieve this, a sequential version in CUDA based on a grid with a single block with only one thread was implemented. The single thread is responsible for calculating all the distances of the N tree nodes for each level.

Figure 10 shows the speedup or degree of parallelization at the Fermi GPU (System 2). It can be observed that it increases as the number of threads that process the information gets higher, since the occupancy of the GPU gets better and it is ensured that a greater number of cores in the GPU are working.

5. Conclusions

GPUs are a very valuable tool, offering relatively high computing power at low cost (€ or $ per flop/s, an interesting and important evaluation performance parameter). However, their use in scientific computing is not yet fully widespread, because there are still some problems for their use by non-specialized programmers. Especially, their programming is quite clumsy, because the programmer must take into account many architecture details that can be safely ignored when programming for a CPU.

As with CPU programming, it is highly desirable the use of specialized libraries (CUBLAS, MAGMA) to relieve the programmer of the task of dealing with very specialized algorithms. Some of these libraries take the approach of using the GPU as support for the CPU (no just as an independent device), using both co-ordinately.

Another problem is that the performance of GPU depends heavily on the problem to be solved (that is, if the algorithm can be casted into a SIMT framework). If the algorithm has many divergent paths, it is very unlikely than a GPU implementation can give good results. Communications between the CPU and the GPU can decrease the throughput of the GPU, especially when the GPU must wait for the results of the CPU, as shown with the K-Best tree search. Generally speaking, algorithms where the different threads must synchronize very often will not profit from the full computing power of the GPU.

Anyway, all the problems for the general use of GPU are being solved through: libraries like MAGMA and CUBLAS, new programming environments such as CUDA, and new hardware improvements such as the new Fermi card. Thus the performance of GPU in Signal Processing applications, as in many other fields, offers promising future possibilities. This will enable the application of known algorithms previously discarded for their large computational cost.

Acknowledgements

This work was financially supported by the Spanish Ministerio de Ciencia e Innovación (Projects TIN2008-06570-C04-02, TEC2009-13741 and CAPAP-H3 TIN2010-12011-E), Universitat Politècnica de València through "Programa de Apoyo a la Investigación y Desarrollo (PAID-05-10)" and Generalitat Valenciana through project PROMETEO/2009/013.

References

[1] GTC 2010 Presentation Archive http://www.nvidia.com/object/gtc2010-presentation-archive.html

[2] NVIDIA, NVIDIA CUDA Programming Guide, Version 3.2, 11/09/2010

[3] Fermi Compute Architecture. www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf

[4] Graphic card ATI Radeon http://www.amd.com/us/products/desktop/graphics/ati-radeon-hd-5000/Pages/ati-radeon-hd-5000.aspx

[5] OpenCL - The open standard for parallel programming of heterogeneous systems http://www.khronos.org/opencl/

[6] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, P. Hanrahan "Larrabee: A Many-Core x86 Architecture for Visual Computing" ACM Transactions on Graphics, Vol. 27, No. 3, Article 18, Publication date: August 2008. pp. 1-15

[7] A. Gonzalez, J. A. Belloch, F. J. Martinez, P. Alonso, V. M. Garcia, E. S. Quintana-Orti, A. Remon, A. M. Vidal, "The Impact of the Multi-core Revolution on Signal Processing", Waves, vol. 2, pp. 74-85, 2010.

[8] A. Gonzalez, J. A. Belloch, G. Piñero, J. Lorente, M. Ferrer, S. Roger, C. Roig, F. J. Martinez, M. de Diego, P. Alonso, V. M. Garcia, E. S. Quintana-Orti, A. Remon, A. M. Vidal, "Application of Multi-core and GPU Architectures on Signal Processing: Case Studies", Waves, vol. 2, pp. 86-96, 2010.

[9] Intel® Microarchitecture Codename Nehalem. http://www.intel.com/technology/architecture-silicon/next-gen/319724.pdf

[10] G. H. Golub and C. F. van Loan. Matrix computations. Johns Hopkins University Press, 1996.

[11] A, Bjork, "Numerical methods for Least Square Problems", Philadelphia PA-SIAM, 1996

[12] 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.

[13] Z. Guo and P. Nilsson, "Algorithm and implementation of the K-Best Sphere Decoding for MIMO detection," IEEE Journal on Selected Areas in Communications, vol.24, no.3, pp. 491–503, March 2006.

[14] E. Anderson, Z. Bai, C. Bishof, J. Demmel, and J. Dongarra. LAPACK User Guide; Second edition. SIAM, 1995.

[15] S. Tomov, R. Nath, P. Du, J. Dongarra, MAGMA version 0.2 Users' Guide, November 2009. http://icl.cs.utk.edu/magma

[16] Grey Ballard, James Demmel, and Andrew Gearhart. "Communication bounds for heterogeneous architectures". Lapack working note 239. February 2011

[17] CULA, tools. CULA Reference Manual. www.culatools.com

[18] Peter N. Glaskowsky "NVIDIA's Fermi: The First Complete GPU Computing Architecture", September 2009

[19] AMD Phenom architecture http://www.amd.com/US/PRODUCTS/DESKTOP/PROCESSORS/PHENOM-II/Pages/phenom-ii.aspx

[20] AMD Opteron 6000 Series Platform Quick Reference Guide. http://www.amd.com/us/Documents/48101C_Opteron__6000_QRG_FINAL.pdf

[21] Intel Polaris http://download.intel.com/pressroom/kits/Teraflops/Teraflops_Research_Chip_Overview.pdf

[22] AMD Fusion http://sites.amd.com/us/Documents/48423B_fusion_whitepaper_WEB.pdf

[23] CUBLAS Library. http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/CUBLAS_Library.pdf

[24] 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.

[25] M. O. Damen, H. E. Gamal, and G. Caire, "On maximum-likelihood detection and the search for the closest lattice point," IEEE Transactions on Information Theory, vol.49, no.10, pp. 2389–2402, October 2003.

[26] MAGMA project website http://icl.cs.utk.edu/magma/

[27] MAGMA QR evaluation performance http://www.cs.utk.edu/~tomov/MAGMA-QR-Fermi.tif

[28] MAGMA LS evaluation performance http://www.cs.utk.edu/~tomov/MAGMA-QR-Solve-Fermi.tif

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 Computing 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.

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.

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 and 2010, 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.

Cristina Yenyxe González García was born in Oviedo, Spain, in 1986. She received her Bachelor in Computer Science from the Universidad de Oviedo, Spain, in 2008. Currently, she is a studying the Master in Parallel and Distributed Computing in Universidad Politécnica de Valencia, Spain. She is also contributing to the project "High Performance Computing on Current Architectures for Problems of Multiple Signal Processing". Her main interest is parallel computing, especially GPU programming.

Carla Ramiro Sánchez was born in Valencia, Spain, in 1984. She received the Engineer Degree in Computer Science and an Technical Engineer Degree in Telematics in 2009, both from the Universitat de Valencia, and the MSc. degree in Parallel and Distributed Computing in 2010, from the Universidad Politécnica de Valencia, Spain. She is working an assistant researcher in the Department of Information Systems and Computation at the Universidad Politécnica de Valencia. Her research focuses on parallelization of signal processing problems on the different cores of a CPU and GPU.