Ramiro Sánchez, C.; Roger Varea, S.; Gonzalez, A.; Almenar Terre, V.; Vidal Maciá, AM. (2013). Multicore implementation of a fixed-complexity tree-search detector for MIMO communications. The Journal of Supercomputing (Online). 65(3):1010-1019. doi:10.1007/s11227-012-0839-x

Noname manuscript No.
(will be inserted by the editor)

Multi-Core Implementation of a Fixed-Complexity Tree-Search Detector for MIMO Communications

Carla Ramiro · Sandra Roger · Alberto Gonzalez · Vicenc Almenar · Antonio M. Vidal

Received: date / Accepted: date

Abstract Multi-core systems allow the efficient implementation of signal processing algorithms for communication systems due to their high parallel processing capabilities. In this paper, we present a high-throughput multi-core implementation of a fixed-complexity tree-search-based detector interesting for MIMO wireless communication systems. Experimental results confirm that this implementation allows to accelerate the data detection stage for different constellation sizes and number of subcarriers.

Keywords MIMO detection · Sphere decoding · multi-core

1 Introduction

Multiple-input multiple-output (MIMO) systems have the ability to increase the maximum transmission rates and the achieved reliability and coverage of current wireless communications without the need for additional bandwidth nor transmit power [1]. MIMO techniques can be also used to enhance the performance of orthogonal frequency division multiplexing (OFDM) systems by exploiting the spatial domain. OFDM is an easy technique to mitigate the effects of inter-symbol interference in frequency selective channels, turning a broadband frequency selective channel into a set of narrowband channels by transmitting data in parallel over the different subcarriers. OFDM combined with MIMO systems, also known as MIMO-OFDM [2], allows transmitting different streams over different subcarriers.

C. Ramiro · A. M. Vidal
Department of Information Systems and Computation
Universidad Politecnica de Valencia
E-mail: cramiro@dsic.upv.es, avidal@dsic.upv.es

S. Roger · A. Gonzalez · V. Almenar
Institute of Telecommunications and Multimedia Applications
Universidad Politecnica de Valencia
E-mail: sanrova@iteam.upv.es, agonzal@dcom.upv.es, valmenar@dcom.upv.es

To boost the data rates of current generation cellular networks, MIMO technologies have been adopted by many wireless standards such as LTE [3], WiMAX and WLAN. The use of MIMO systems, however, complicates the receiver stage, which has the task of processing the received mixture of signals affected by the channel in order to recover the transmitted data with the highest reliability. In fact, if nearly optimal detection is desired, this stage becomes often the most computationally expensive within a MIMO system. Furthermore, scalability in the number of subcarriers per MIMO-OFDM symbol and in the system size are key factors in wireless standards [3]. All the above reasons motivate the search for high-throughput receiver implementations capable to be reconfigured and scalable with the system parameters.

The practical implementation of MIMO receiver schemes and software-defined-radio (SDR) platforms have been traditionally developed using digital signal processors (DSP) [4], field programmable gate arrays (FPGA) or application-specific integrated circuits (ASIC) [5][6]. Recently, the use of the last generation multi-core CPUs and GPUs has become attractive for the efficient implementation of parallel signal processing algorithms with high computation requirements, such as the scheme reported in [7] and [12] respectively.

In this work we demonstrate the usefulness of multi-core CPU to develop high-throughput implementations of signal processing algorithms for communication systems. The proposed approach ensures high-throughput nearly optimal detection performance in MIMO-OFDM systems.

2 System model and MIMO detection

Let us consider a MIMO system with n_T transmit antennas, n_R receive antennas ($n_R \geq n_T$) and a certain signal-to-noise ratio (see Fig. 1). The input data stream is split equally into the n_T transmit antennas and sent simultaneously through the channel, thus overlapping in time and frequency. The baseband equivalent model for this system is given by

$$\mathbf{x} = \mathbf{H}\mathbf{s} + \mathbf{v}, \tag{1}$$

where $\mathbf{s}$ represents the transmitted signal vector composed of the elements resulting of mapping sets of information bits to symbols belonging to a certain alphabet of complex numbers (constellation) $\mathbf{\Omega}$ of size M, such as Quadrature Amplitude Modulation (QAM). Vector $\mathbf{x}$ in (1) denotes the received symbol vector, and $\mathbf{v}$ is a complex additive white Gaussian noise vector. The Rayleigh fading channel matrix $\mathbf{H}$ is assumed to be known at the receiver and is formed by $n_R \times n_T$ complex-valued elements, h_{ij}, which represent the fading gain from the j-th transmit antenna to the i-th receive antenna. It is important to note that, in this work, a block fading channel is considered, which means that it is possible to transmit a long data frame without estimating a new channel matrix. Since MIMO-OFDM transmission is considered, the system model holds for the transmission through a single subcarrier out of the N_c subcarriers used in the system.

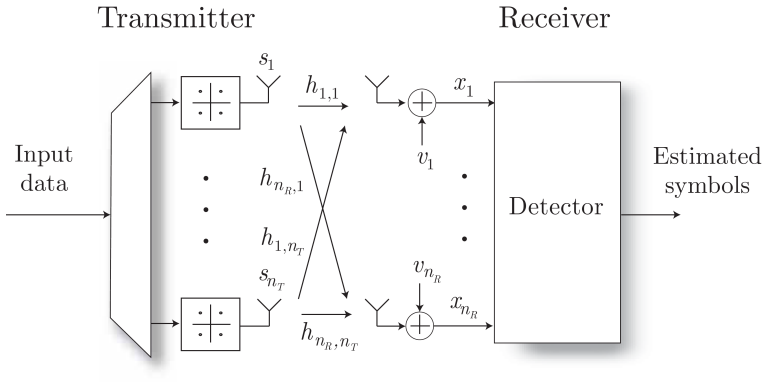

Fig. 1 Block diagram of a MIMO system.

Given the received signal $\mathbf{x}$, the detection problem in MIMO systems consists in determining the transmitted vector $\hat{\mathbf{s}}$ with the highest a posteriori probability [1]. In practice, this detection problem is carried out by solving the following least squares problem

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

which can be solved by an exhaustive search over a total n_T-dimensional lattice points $\mathbf{s}$. The solution of (2) is known as the *maximum-likelihood* (ML) estimate. This implementation is cumbersome for practical systems; however, its complexity can be substantially reduced by means of tree-search detection methods such as the fixed-complexity sphere decoder (FSD) [8] or many other sphere decoding (SD) methods [9].

If a QR factorization of the channel matrix ($\mathbf{H} = \mathbf{Q}\mathbf{R}$) is employed, the problem (2) can be transformed into an equivalent one that can be solved using a tree structure [10]. $\mathbf{Q}$ is an unitary matrix ($\mathbf{Q}\mathbf{Q}^H = \mathbf{I}$) and $\mathbf{R}$ can be decomposed into an upper triangular $n_T \times n_T$ matrix, denoted by $\mathbf{R}'$, and a $(n_R - n_T) \times n_T$ matrix of zeroes. For the sake of simplicity, a system with $n_T = n_R$ is assumed.

In case of multiplying (2) by $\mathbf{Q}^H$ and calling $\mathbf{y} = \mathbf{Q}^H\mathbf{x}$, the problem (2) can be equivalently expressed as

$$\hat{\mathbf{s}} = \arg \min_{\mathbf{s} \in \Omega^{n_T}} \{||\mathbf{y} - \mathbf{R}\mathbf{s}||^2\}. \tag{3}$$

In the next section, we will see how the FSD method uses the expression (3) to solve the detection problem via a tree-search is described.

3 Fixed-Complexity Sphere Decoder

In [8] the authors proposed a MIMO detection strategy intended to overcome the two main problems of other tree-search methods from an implementation point of view: their variable complexity and their sequential nature. This

algorithm was called fixed-complexity SD (FSD) and combines a preprocessing stage followed by a predetermined tree-search composed of two different stages: a full expansion of the tree (FE) in the first (highest) T levels and a single-path expansion (SE) in the remaining tree-levels $n_T - T$ [8].

A search-tree is built containing all the candidate lattice points associated to the problem to be solved. The tree must have as many levels as transmit antennas and each symbol value is represented by a tree node. The tree-paths are built by connecting nodes and stand for candidate solutions. For instance, a tree-path containing selected symbols from the root up to level i has the form $S^{(i)} = [s_i, s_{i+1}, \ldots, s_{n_T}]^T$.

The symbols are detected following a specific ordering also proposed by the authors in [8] which is based on the following reasoning: if all the possibilities are searched in one level, the robustness of the signal at such level is not relevant to the final performance. Therefore, the signals that suffer the largest noise amplification are placed at the levels where a FE is performed. On the other hand, the signals that suffer the smallest noise amplification are placed at the levels associated to the SE.

The FSD ordering iteratively orders the n_T columns of the channel matrix $\mathbf{H}$. At the i-th iteration only those components still to be detected are considered. The particular steps carried out for each iteration are the following. First, the pseudoinverse of the matrix containing only the columns of the indexes not selected yet ($\mathbf{H}_i$) is computed:

$$\mathbf{H}_i^+ = (\mathbf{H}_i^H \mathbf{H}_i)^{-1} \mathbf{H}_i^H, \qquad i = n_T, \ldots, 1. \tag{4}$$

Then, if a symbol for the FE is searched, the index that satisfies the following is selected $k = \arg\max \|\mathbf{H}_i^+\|^2$, otherwise, the selected index must fulfill $k = \arg\min \|\mathbf{H}_i^+\|^2$.

Although the FSD does not guarantee to find the ML estimate, it achieves the same average performance as the ML detector provided $T \geq \sqrt{n_T} - 1$, as shown in [8].

Figure 2 shows the search tree of the FSD algorithm for the case with $n_T = 4$ ($T = 1$) and symbols belonging to a Quadrature Phase-Shift Keying (QPSK) constellation. At the FE stage, for each survivor path, all the possible values of the constellation are assigned to the symbol at the current level. The SE stage starts from each retained path and obtains the remaining unknowns (those in the lowest $n_T - T$ tree-levels) using successive interference cancellation (SIC) as follows:

$$\hat{\mathrm{s}}_i = \mathcal{Q}\left\{ y_i - \sum_{l=i+1}^{n_T} R_{i,l} \hat{s}_l \right\}, \qquad i = n_T - T, \ldots, 1. \tag{5}$$

where $\mathcal{Q}(\cdot)$ denotes quantization to the nearest constellation point. After the set of candidate solutions is built, a sorting stage is necessary to determine the path with the lowest Euclidean distance to the received vector. These path

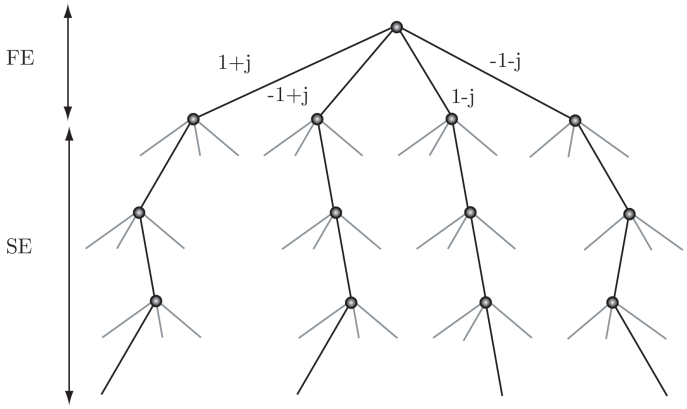

Fig. 2 Decoding tree of the FSD algorithm for a 4×4 MIMO system with QPSK symbols.

distances are calculated iteratively from the partial Euclidean distance (PED) at each level, which for level i equals:

$$e_i(S^{(i)}) = y_i - \sum_{j=i}^{n_T} R_{i,j}\hat{s}_j. \tag{6}$$

Then, the accumulated PED for each path is directly

$$d_i(S^{(i)}) = d_{i+1}(S^{(i+1)}) + |e_i(S^{(i)})|^2, \tag{7}$$

assuming that $d_{n_T+1}(S^{(n_T+1)}) = 0$.

Since all the SIC problems that compose the SE stage are totally independent among themselves, they can be carried out in parallel, which makes the FSD algorithm very suitable for multi-core implementation.

4 FSD multi-core Implementation

During the last few years, the leading hardware manufacturers have been focused in developing up to 12 cores processors. Their purpose is to get maximum performance with minimal consumption using hyper-threading and other technologies. In this work we made use of a high performance compute with two Intel Xeon X5680 processors running at 3.33 GHz. Each of them is an hexacore processor supporting hyper-threading (the system has 24 virtual processors), with 12 MB of cache memory and 96 GB of GDDR3 main memory.

For the parallelization of the FSD algorithm we assumed a MIMD computer (i.e. *multiple instruction, multiple data*) with shared memory. This system has p processors which share a common central memory. We used the OpenMP application programming interface [11], which supports multi-platform shared memory multiprocessing programming in C language.

The proposed multi-core implementation is composed of two differentiated parts. On the one hand, the preprocessing stage to reorder the channel matrix

and to next carry out the QR decomposition of it is performed at the beginning and only every time the channel changes. As a block-fading channel is considered in this work, it remains constant for the transmission of a whole time slot (7 OFDM symbols). On the other hand, the tree-search-based detection is performed in parallel after grouping all the symbols contained in the time slot.

As it was already described, the FSD ordering requires the computation of a pseudoinverse matrix in every step. This calculation will be performed efficiently by solving the two linear systems that are next presented.

The pseudoinverse matrix to be computed at step i is $\mathbf{G} = \mathbf{H}_i^+$. Calling $\widetilde{\mathbf{H}}_i = (\mathbf{H}_i^H \mathbf{H}_i)$ and with $\widetilde{\mathbf{H}}_i = \widetilde{\mathbf{Q}}_i \widetilde{\mathbf{R}}_i$, if $\mathbf{G}$ is multiplied by $\widetilde{\mathbf{H}}_i$, it can be equivalently expressed as:

$$\widetilde{\mathbf{H}}_i \mathbf{G} = \widetilde{\mathbf{Q}}_i \widetilde{\mathbf{R}}_i \mathbf{G} = \mathbf{H}_i^H. \tag{8}$$

Next, (8) can be multiplied by $\widetilde{\mathbf{Q}}_i^H$ resulting in

$$\widetilde{\mathbf{Q}}_i^H \widetilde{\mathbf{Q}}_i \widetilde{\mathbf{R}}_i \mathbf{G} = \widetilde{\mathbf{R}}_i \mathbf{G} = \widetilde{\mathbf{Q}}_i^H \mathbf{H}_i^H. \tag{9}$$

To solve (9), first an auxiliary matrix $\mathbf{J}_i$ is computed as:

$$\mathbf{J}_i = \widetilde{\mathbf{Q}}_i^H \mathbf{H}_i^H, \tag{10}$$

then, solving the following upper triangular system gives matrix $\mathbf{G}$ as a result:

$$\widetilde{\mathbf{R}}_i \mathbf{G} = \mathbf{J}_i. \tag{11}$$

Algorithm 1 shows the code description to carry out the ordering of the N_c different channel matrices necessary to detect the symbols in one time slot and also the QR decomposition of the reordered channel matrix. Once the selected ordering is obtained, the MIMO channel matrices $\mathbf{H}$ are transformed via a matrix $\mathbf{P}$ into a new channel matrix $\mathbf{HP}$. To keep the system model unaltered, the detected symbol vector is also reordered as $\mathbf{P}^{-1}\mathbf{s}$. Note that these calculations are distributed among all threads (cores) using the OpenMP clause *schedule dynamic*.

Algorithm 2 contains the pseudo-code related to the tree-search detection stage. The calculation of all the branches of the FSD for N_c different channel matrices are again distributed among all the cores, as done in the Algorithm 1. Given that the calculation of all the subcarriers can be done independently, because the FSD algorithm is applied performed on different channel matrices and symbols received, synchronization among the threads is not required.

Algorithm 1 Parallel calculations carried out at the FSD preprocessing stage with CPU cores

IN PARALLEL: Distribute among p threads the N_c subcarriers,
Get $\mathbf{H}$ associated to the current subcarrier from global memory
$\Delta = \{1, 2, \cdots, n_T\}$
$dis_{1:n_T} = [1, 1, \cdots, M]$
$\mathbf{P} = zeros(n_T, n_T)$
for $i = n_T, \ldots, 1$ **do**
  $\mathbf{H}^{(i)} = \mathbf{H}_{:,\Delta}$
  Obtain $\mathbf{G}$ solving Eqs. (10) and (11)
  $d_{max} = 0$
  $d_{min} = 1e6$
  for $j = 1, \ldots, length(\Delta)$ **do**
    $norm_j = \|G_{j,:}\|^2$
    if $dis_i == M$ and $norm_j > d_{max}$ **then**
      $d_{max} = norm_j$
      $k = \Delta_j$
    else if $dis_i \sim= M$ and $norm_j < d_{min}$ **then**
      $d_{min} = norm_j$
      $k = \Delta_j$
    end if
  end for
  $P_{k,i} = 1$
  $\Delta = \Delta - \{\Delta_j\}$
end for
Permute columns of matrix $\mathbf{H}$ with $\mathbf{P}$
$\mathbf{H} = \mathbf{QR}$
Compute $\mathbf{y} = \mathbf{Q}^H\mathbf{x}$ for the 7 symbols in the time slot
END PARALLEL

Algorithm 2 Parallel calculation of all branches of the hard-output FSD of qth subcarrier

IN PARALLEL: Distribute among p threads the N_c subcarriers,
for $j = 1, \ldots, 7$ symbols int the time slot **do**
  for $i = 1, \ldots, M$ **do**
    Assign $s_{n_T} = \Omega_i$,
    Get $\mathbf{R_q}$ and $\mathbf{y_{q,j}}$ associated to the current subcarrier,
    Compute the PED $d_{n_T}(S_q^{(n_T)})$ with equations (6-7)
    for $k = n_T - 1, \ldots, 1$ **do**
      Compute the kth symbol using SIC (5),
      Update path distance $d_k(S_q^{(k)})$ using (7),
    end for
    Get minimal distance of all paths
  end for
end for
END PARALLEL

5 Experimental Results

We consider a 4×4 MIMO system with QPSK, 16-QAM and 64-QAM symbol alphabets, these values correspond to those used in the standards of current mobile and wireless communications, such as LTE. According to the LTE standard specifications, a 0.5 ms time slot is composed of 7 MIMO-OFDM

symbols plus their respective cyclic prefixes [2]. Furthermore, the performances with the different N_c values reported in the LTE standard, i.e. $N_c = 150$, 300, 600, 900, 1200, are investigated. Two different performance measures were considered to evaluate the proposed implementation:

- *Speedup*, which is defined as the ratio between the computational time resulting of executing the algorithm sequentially on a single CPU core and the time to execute the same algorithm using a multi-core approach.
- *Throughput*, which is defined as the number of processed information bits per second. The throughput evaluation exposes whether a given implementation guarantees the real-time requirements of a certain wireless standard.

For all the experiments, the algorithm was executed varying the number of active cores ranging from 1 to 24 (hyper-threading). The best results were selected in every case (in most cases those obtained using either 16 or 24 cores). In general, for numerically intensive computations, hyper-threading does not improve performance overly, since we have not really 24 physical processors.

Table 1 collects the throughput and runtime results of the proposed multi-core FSD implementation. Note that, for the QPSK case, the proposed FSD multi-core implementation can process a whole slot before having received the following one (i.e. in $\leq$ 0.5 ms) for all the considered N_c values. Thus, the QPSK configuration allows real-time processing. For the 16-QAM case, the runtime is less than 0.5 ms only for either $N_c = 150$ or $N_c = 300$. Finally, when 64-QAM symbols are used, none of the configurations meets the real-time goal. Therefore, the FSD implementation requires further optimizations to meet real-time for those configurations where the runtime is above 0.5 ms.

Table 1 Throughput and execution time of the proposed multi-core implementation assuming a 4×4 MIMO system with for different configurations compared to FSD parallelized on GPU results in [12].

	Throughput(Mbps)/Runtime(ms)		
	QPSK	16-QAM	64-QAM
FSD ($N_c = 150$)	131.25 / 0.064	54.55 / 0.308	28.44 / 0.886
FSD ($N_c = 300$)	173.20 / 0.097	80.19 / 0.419	29.18 / 1.727
FSD ($N_c = 600$)	182.61 / 0.184	94.78 / 0.709	31.18 / 3.233
FSD ($N_c = 900$)	182.6 / 0.276	98.25 / 1.026	31.07 / 4.866
FSD ($N_c = 1200$)	183.61 / 0.366	90.26 / 1.489	31.45 / 6.410
FSD [12] ($N_c = 150$)	182.61 / 0.046	311.11 / 0.054	115.07 / 0.219
FSD [12] ($N_c = 300$)	311.11 / 0.054	436.36 / 0.077	123.23 / 0.409
FSD [12] ($N_c = 600$)	430.77 / 0.078	533.33 / 0.126	130.57 / 0.772
FSD [12] ($N_c = 900$)	494.12 / 0.102	579.31 / 0.174	131.25 / 1.15
FSD [12] ($N_c = 1200$)	537.60 / 0.125	610.91 / 0.220	132.81 / 1.52

Figure 3 shows the speedup results for different values of N_c. The speedup for the preprocessing stage is common for the constellations as it only depends on the channel matrix. Thus, a single curve regarding preprocessing speedup is shown in the figure. However, the speedup of the FSD tree-search

stage depends on the channel matrix and also on the constellation size, thus, independent curves are represented for each constellation used.

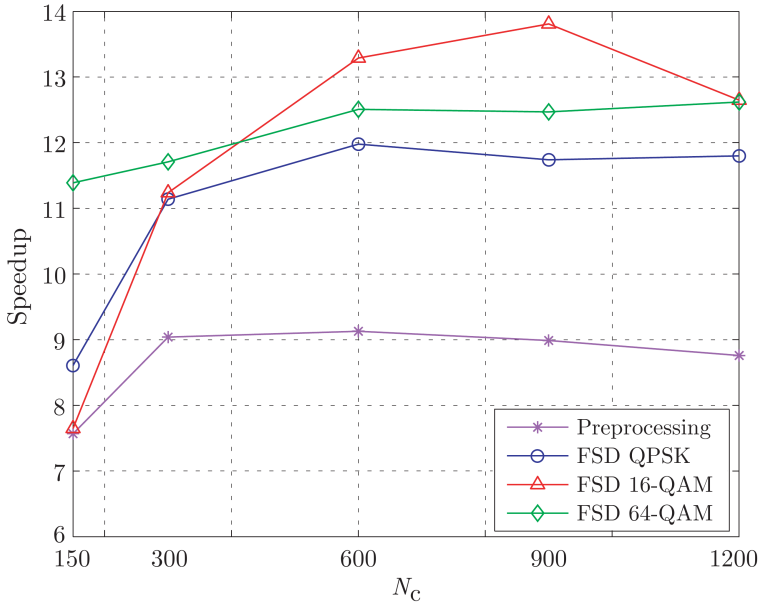

Fig. 3 Speedup for the proposed FSD implementation with different constellations and number of subcarriers in a 4×4 MIMO system.

It can be seen that, generally, the speedup increases with N_c, showing that the use of multi-core is very promising for high dimensional constellations and/or LTE configurations managing a large amount of subcarriers. Furthermore, for the FSD tree-search stage, it can be observed that the higher the constellation size, the higher the achieved speedup. The 64-QAM case is an exception, as it reaches higher speedup than QPSK configuration but lower speedup than the 16-QAM one. This non-expected result may be caused by the need for a higher amount of memory in the 64-QAM together with the associated cost of memory accesses.

The speedup of the preprocessing stage is a bit lower than that of the FSD stage. The main reason can be that, at this stage, the threads are working with big shared variables. These shared variables stored the data for the N_c subcarriers ($\mathbf{H}$, $\mathbf{R}$, $\mathbf{P}$, $\mathbf{y}$). The cost for different cores to access blocks in memory can influence the efficiency of the algorithm, because a thread can be running on a CPU and their access to some shared variables are realized on the memory of another CPU. However, there is a library Unified Parallel C (UPC) [13], which aims to solve this performance problem in shared memory multiprocessors, which defines the physical association between shared data and UPC threads, so that we can store the data in physical memory of the

CPU where the thread is running, giving better performance in shared memory accesses.

6 Conclusions

This work showed the potential of parallel processing to cope with the costly signal processing algorithms necessary to carry out MIMO wireless communication. An efficient multi-core implementation of a fixed-complexity MIMO detector was here presented. The experimental results indicate that the proposed implementation considerably reduces the execution time required to perform data detection with respect to a single-core (sequential) CPU implementation of the same detection algorithm. Furthermore, the runtime of the proposed approach fulfills the requirements of current wireless communication standards for some configurations. Therefore, the high throughput and ability to reconfigure features clearly evidence that multi-core systems are meaningful to develop versatile and low-cost SDR platforms.

Acknowledgements This work was supported by the TEC2009-13741 project of the Spanish Ministry of Science, by the PROMETEO/2009/013 project and ACOMP/2012/076 of the Generalitat Valenciana, and the Vicerrectorado de Investigación de la UPV through Programa de Apoyo a la Investigación y desarrollo (PAID-05-11-2898)

References

1. 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, February 2004.
2. M. Jiang and L. Hanzo, *Multiuser MIMO-OFDM for Next-Generation Wireless Systems*, Proceedings of the IEEE, vol. 95, no. 7, pp. 1430-1469, July 2007.
3. 3GPP TS 36.201, V10.0.0, *Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Layer - General Description*, December 2010.
4. Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti and K. Flautner, *SODA: A High-Performance DSP Architecture for Software-Defined Radio*, Micro, IEEE, vol. 27, no. 1, pp.114-123, Jan.-Feb. 2007.
5. C.-H. Yang and D. Markovic, *A Multi-Core Sphere Decoder VLSI Architecture for MIMO Communications*, Global Telecommunications Conference, pp.1-6, November 2008.
6. D. Wu, J. Eilert and D. Liu, *Implementation of A High-Speed MIMO Soft-Output Symbol Detector for Software Defined Radio*, Journal of Signal Processing Systems, vol. 63, no. 1, pp. 27-37, April 2011.
7. K. Tan, H. Liu, J. Zhang, Y. Zhang, J. Fang and G.M. Voelker., *Sora: high-performance software radio using general-purpose multi-core processors*, Commununications of the ACM, vol. 54, no.1, January 2011.
8. L.G. Barbero and J.S. Thompson, *Fixing the Complexity of the Sphere Decoder for MIMO Detection*, IEEE Transactions on Wireless Communications, vol. 7, no. 6, pp. 2131-2142, June 2008.
9. B. Hassibi and H. Vikalo, *On Sphere Decoding algorithm. Part I, The expected complexity*, IEEE Transactions on Signal Processing, vol. 54, no. 5, pp. 2806-2818, August 2005.
10. E. Agrell, T. Eriksson, A. Vardy and K. Zeger, *Closest point search in lattices*, IEEE Transactions on Information Theory, vol. 48, no. 8, pp. 2201-2214, August 2002.

11. OpenMP v3.0, *http://www.openmp.org/mp-documents/spec30.pdf*, May 2008.
12. S. Roger, C. Ramiro, A. Gonzalez, V. Almenar and A.M. Vidal, *An efficient GPU implementation of fixed-complexity sphere decoders for MIMO wireless systems* , Integrated Computer-Aided Engineering, In Press.
13. Unified Parallel C, *http://upc.lbl.gov/*