J Supercomput (2015) 71:751–760
DOI 10.1007/s11227-014-1328-1

MIMOPack: a high-performance computing library for MIMO communication systems

Carla Ramiro · Antonio M. Vidal · Alberto Gonzalez

Published online: 5 December 2014

Abstract This paper presents MIMOPack, a set of optimized functions to perform some of the most complex stages in multiple-input multiple-output (MIMO) communication systems such as channel coding, preprocessing, precoding and detection. These functions are optimized to be run in a wide range of architectures increasing the portability of scientific codes between different computing environments. MIMOPack aims to become a useful library for the research community facilitating to the programmer the development of adaptable parallel applications and also to speed up simulation platforms used to assess different technologies proposed by several companies involved in standarization processes.

Keywords HPC library · GPU · Multi-core · CUDA · MIMO

1 Introduction

Multiple-input multiple-output (MIMO) systems have a high impact in the current wireless communications, since they allow increase in the realiability, coverage and transmission rates without the need for extra-bandwidth or power cost [1]. To boost

C. Ramiro (✉) · A. M. Vidal
Department of Information Systems and Computation,
Universitat Politècnica de València, Valencia, Spain
e-mail: cramiro@dsic.upv.es

A. M. Vidal
e-mail: avidal@dsic.upv.es

A. Gonzalez
Institute of Telecommunications and Multimedia Applications,
Universitat Politècnica de València, Valencia, Spain
e-mail: agonzal@dcom.upv.es

the data rates of current generation cellular networks, MIMO technologies have been adopted by many wireless standards such as long-term evolution (LTE), worldwide interoperability for microwave access (WiMAX), wireless local area network (WLAN) and also by broadband standards such as digital video broadcasting next-generation handheld (DVB-NGH). The influence of the new handheld devices such as smart phones and tablets is also important, since it has caused the drastic growth of mobile multimedia services. Hence, MIMO surely will become an imperative technology of wireless communication systems to increase the data traffic capacity.

However, the use of MIMO technologies involves an increment of the detection process complexity. The detector is present in the receiver side and is the responsible for recover the transmitted signals (which are affected by the channel fluctuations) with the maximum reliability. This step becomes in many cases the most complex stage in the communication. Another important factor that affects the performance of a MIMO system is the number of transmit and receive antenna, because as the system grows the communication data processing becomes more complicated. Although the number of antennas currently allowed in the standards is not large, it is expected that more than 100 transmit antennas could be used in the near future [2]. All the above reasons motivate the search for high-throughput versatile receiver implementations capable to be reconfigured and scalable with the system parameters.

Practical implementation of MIMO receiver schemes and software-defined radio (SDR) platforms has been traditionally developed using digital signal processors (DSP) [3], field programmable gate arrays (FPGA) or application-specific integrated circuits (ASIC) [4,5]. The last advances in computational hardware have allowed the development of high-throughput implementations. In last years, the number of scientific contributions and research projects related to the use of high-performance computing (HPC) systems has significantly increased. This phenomenon has occurred in almost all engineering fields that require intensive computing, and signal processing is not an exception.

The use of the last generation of high-performance computing (HPC) systems such as multi-core CPUs and graphic processing units (GPUs) has become attractive for the efficient implementation of parallel signal processing algorithms with high computational requirements, such as the scheme reported in [6], high-throughput MIMO detectors [7,8] and fast low-density parity check (LDPC) decoders [9]. Currently, the simultaneous use of different types of architectures (GPUs and CPU) on a same heterogeneous system enjoys great popularity and is currently used, for example, by numerical linear algebra libraries as MAGMA [10] or CULA [11].

2 Motivation

The use of the high-performance computing systems not only brings big benefits, but also poses big challenges. These systems introduce asymmetries and heterogeneities that complicate the development of efficient algorithms. In the recent years, a large variety of machine architectures have appeared, in view of this situation scientients of the research community are obliged to write the codes in different programming languages and consider many details of the architecture to use efficiently the whole target system. Therefore, this high-performance computing library is essential to facilitate the implementation of scientific codes on a widespread range of architectures. Furthermore, there are several important companies involved in the development of new wireless and broadband standards. Each company proposes technologies to satisfy the functional requirements of the new standards. In many cases, simulation is the only way to develop and test these proposals but these simulations often involve high-computational burden, and require weeks or even months to be completed. Thereby, this library may allow the launch of large simulations, opening the door for researchers to analyze its technologies faster than their conventional simulation.

In this sense the HPC libraries became valuable tools for specialists of a particular field, since it facilitates the development of scientific codes. Some software companies have already released to the market various libraries with extensive backgrounds and acceptance, most of them in numerical linear algebra (e.g. BLAS, LAPACK). However, there are few tools or high performance libraries available for communication systems design. For example, Communications System Toolbox [12] provides algorithms for designing, simulating and analyzing communications systems. Although this software is excellent and widely used by the scientific community, nowadays just a small set of functions are prepared to use parallel computing with GPUs. Other library is IT++ [13], which is a C++ library of mathematical, signal processing and communication classes and functions. IT++ makes an extensive use of existing open-source or commercial libraries for increased functionality, speed and accuracy. In particular BLAS, LAPACK and FFTW libraries can be addressed. However, this library is oriented to its exclusive use on multicore machines. It does not support the use of GPUs.

As mentioned, existent tools do not provide features to ease the development of adaptable parallel applications in accordance with the architecture of the executing platform, so we have made a special effort to develop a high-performance library for MIMO communication systems called MIMOPack. The library aims to implement efficiently, using parallel computing, a set of functions to perform some of the critical stages in MIMO communication systems. One of the main features of this library is that it can be used with the last generation of machine architectures (e.g GPUs and multicore) to fully exploit its computational capacity and reduce the response time of costly and complex problems.

3 MIMOPack software package

The library is written in C language and is composed by several modules. Figure 1 illustrates the basic simulation chain through the library modules. The interface of the functions is common to all environments to ease its use, regardless of the machine where it was executed. This feature increases the portability of codes between different computing platforms. Nowadays, it supports the execution of its routines on a sequential/multicore processor and GPU/Multi-GPU devices. In future releases are expected to enable the use of the Intel Xeon Phi coprocessor.

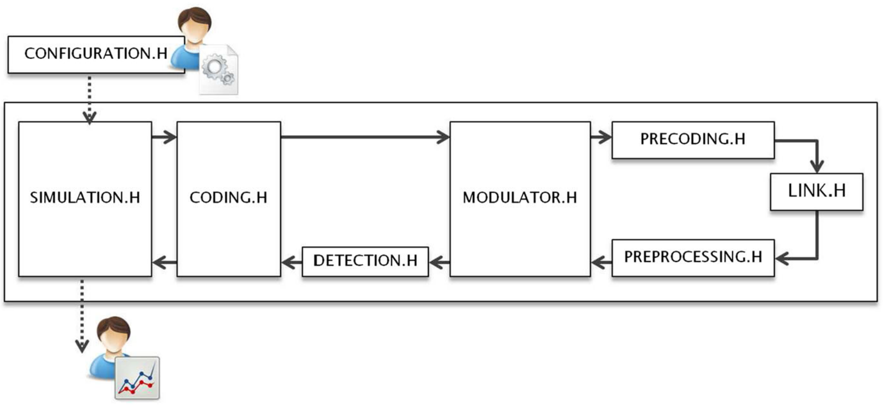

Fig. 1 Simulation chain through the MIMOPack library modules

The library is continuously growing, the updated release collects a set of functions divided in four blocks (most of them ready to be called from MATLAB through MEX-Files):

- Hard-output detection: ML-exhaustive (MLE), zero forcing SIC (ZF-SIC), hard fixed sphere decoder (HFSD), Kbest decoder (KBEST), Schnorr Euchner sphere decoder (SDSE), automatic sphere decoder (ASD). For some performance results see [14–16].
- Soft-output detection: Soft fixed sphere decoder, fully parallel fixed-complexity soft-output detector, max-log. For some performance results see [14–17].
- Precoding: Zero-forcing precoding, Tomlinson–Harashima, lattice reduction-aided Tomlinson–Harashima, lattice reduction aided, enhanced lattice reduction aided. For some performance results see [18,19].
- Error control coding: LDPC decoder [20].

Also other kind of auxiliary functions such as QAM modulators (modulator.h), AWGN channel functions (link.h), preprocessing algorithms (preprocessing.h) and several mathematical functions are included in the library.

3.1 MIMOPack configuration

One of the most important features of MIMOPack is the configurability. The library allows to configure for each simulation the following MIMO systems parameters:

- *nT*: Number of transmitter antennas.
- *nR*: Number of receiver antennas.
- *snr*: Signal to noise ratio (dB).
- *M*: Modulation (constellation size).
- *Nc*: Number of signal vectors to be transmitted.
- *Lch*: Variation of the channel, is the number of transmitted signal vectors with the same channel matrix.

Furthermore, the library allows the user to assign the type and number of resources to use during the execution. For a given C source code, the user may specify a different distribution of the computational resources for each function by the configuration of the *mmp_config* structure which contains the following parameters:

– *omp_threads* is the number of OpenMP threads.
– *n_gpus* is the number of GPUs. The library automatically selects those with more CUDA cores on the detected devices.
– *workload_gpu* is the percentage of workload that will be executed among the *n_gpus*. The remaining work will be computed by *omp_threads* threads.
– *use_mkl* indicates if MKL library will be used in some parts of the algorithms.

Two commands are currently provided by the package to allow the user the correct configuration of the environment: *set_mmp_config* and *check_config*. The first function is used to fix parameters of the *mmp_config* structure. To ensure the proper configuration, the library implements the function *check_config*, which checks the properties of the execution platform and indicates the setup incompatibilities, if any. This routines can be called from a C source code as follows:

```
mmp_config set_mmp_config(int num_gpus, int omp_threads,
int workload_gpu, bool use_mkl);
void check_config(mmp_config config);
```

A calling example could be the following. In this example, we are using a configuration with 80 % of workload for two GPUs and 20 % of workload for 12 OpenMP threads without any MKL call.

```
mmp_config my_configuration;
my_configuration = set_mmp_config(2, 12, 80, false);
check_config(my_configuration);
```

3.2 Heterogenous mode

The library allows the execution of several functions in a heterogeneous mode to fully exploit the computational capacity. The workload can be split between the different types of devices on the target machine (GPU or CPU). Normally, the user will try to simulate a large number of transmitted signals, then the dispatcher splits the amount of signals among the different resources. Using the parameter *workload_gpu* defined in the Sect. 3.1 we can specify the number of signals that each device must perform.

Figure 2 shows a simple example for the detection of 10 signal vectors, denoted as $x^{(i)}$ where i is the index for a given signal. The configuration is 40 % for 4 CPU threads (4 signals) and 60 % for 2 GPUs (6 signals). Initially, the dispatcher launches 2 threads. One of the threads will distribute the load percentage among *omp_threads* threads; to carry out this distribution the option *omp_nested* must be activated allowing nested parallelism. The other thread is in charge of controlling and creating many threads as GPUs have selected with the parameter *n_gpus* seen in previous section.

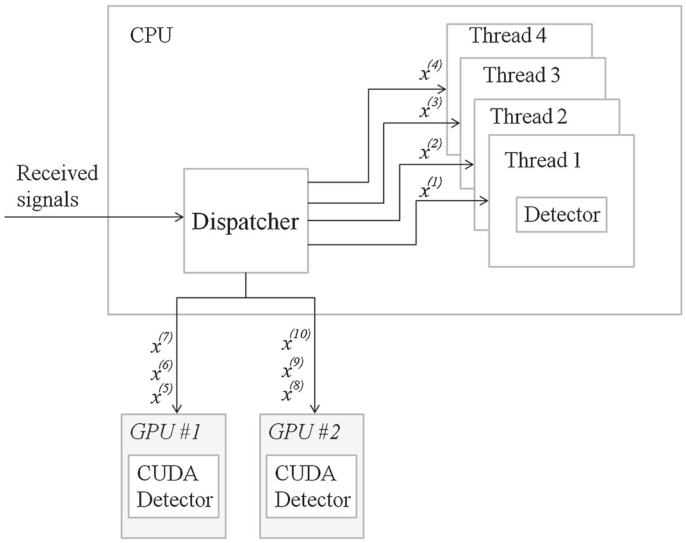

Fig. 2 Example of MIMO detection with heterogeneous mode

3.3 Example of simulation with MIMOPack

Let us consider a MIMO system with nT transmitter and nR receiver antennas and a certain signal-to-noise ratio (SNR). The input data stream is split equally into nT transmit antennas. The baseband equivalent model for this system is given by

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

where **x** represents the transmitted signal vector composed of the elements resulting of mapping sets of information bits to symbols belonging to a certain constellation Ω of size M. Vector **y** denotes the received symbol vector, and **v** is a complex additive white Gaussian noise vector. The detection problem in MIMO systems consists in determining the transmitted vector **xm** with the highest reliability. In practice, the detection problem is carried out by solving the following least squares problem

$$\mathbf{xm} = \arg \min_{\mathbf{x} \in \Omega^{nT}} \|\mathbf{y} - \mathbf{H}\mathbf{x}\|^2. \tag{2}$$

To exemplify the use of the library, we show below an example for a simulation of the transmission of Nc signals in the system explained above.

```
mmp_conf my_conf;
my_conf = set_mmp_config(2, 32, false, 80);
mmp_modulator my_mod;
my_mod = set_mmp_modulator(M, nT);
mmp_simulation my_sim;
my_sim = set_mmp_simulation(M, nR, nT, Nc, Lch, snr,
     my_mod);
7.  generate_bits(bits, Nc*my_mod.k*nT);
8.  qam_mapper_demux(my_mod, bits, Nc*my_mod.k*nT, x_r,
     x_i);
9.  rand_channel_values(H_r, H_i, Lch, Nc, nT, nR);
10. pass_channel(H_r, H_i, x_r, x_i, y_r, y_i, snr, Nc,
     Lch, nT, nR);
11. init_timer(my_sim);
12. dt_mle(my_conf, my_sim, ped, xm_r, xm_i, H_r, H_i,
     y_r, y_i);
13. stop_timer(my_sim);
14. save_simulation(my_sim, x_r, x_i, xm_r, xm_i);
15. printf(Time = \%f, my_sim.etime);
16. printf(Time = \%f, my_sim.ber);
```

As we can see, there is an initial stage where some parameters of the platform, modulator and simulation are configured (see lines 1–6). After that, we need to generate the random bits which simulate the transmission of a service. Next, these bits are multiplexed and mapped into constellations symbols. Also it is necessary to generate the random channel values to simulate the conditions of the link through which they will be sent. Then, we initialize the timer to assess the performance of the detector (MLE detector in line 12) with the configuration selected in the line 2. Finally, the user must save the simulation results to calculate some statistics (e.g. execution time, bit error rate, symbol error rate or throughput).

4 Performance evaluation

To assess the performance of our library, we have evaluated the execution times of a set of hard-output detectors. Clearly, there are several methods and versions in the state of the art devised to recover the information in a MIMO system, but only a selection of these have been evaluated in this paper. This selection is formed by detectors with mixed complexities and performances intended to cover multiple use cases with different MIMO scenarios and channel conditions. These detectors can be grouped in two main groups: ML and non-ML detectors. The ML detectors (MLE and SDSE) are optimum in terms of bit error rate but at the expense of increased complexity. On the other hand, non-ML detectors (ZF-SIC and KBEST) are simplest but the reliability of the systems decreases when using this kind of detectors to obtain an approximation of the detected symbols [21].

The tests are executed on a platform with one Nvidia Tesla K20Xm GPU with 14 SM, each SM including 192 cores. The core frequency is 0.73 GHz. The GPU has 5GB of GDDR5 global memory and 48 kB of shared memory per block. The installed CUDA toolkit is 5.5. The Nvidia card is mounted on a PC with two Intel Xeon CPU E5-2697 at 2.70 GHz with 12 cores and hyperthreading activated. We consider a simple simulation example of a MIMO system with 6 transmit and receive antennas, 16-QAM symbol alphabet and a channel SNR of 5dB. The speedup is defined as the ratio between the computational time resulting of executing the simulation of 50,000

Table 1 Runtime and bit error rate of hard-output MIMOPack detectors with different library configurations

Detector	Runtime (s)			BER
	Sequential	32 threads	GPU	
MLE	1.28×10^5	7.41×10^3	5.69×10^3	0.083205
SDSE	34.44	2.39	7.46	0.083205
ZFSIC	0.32×10^{-1}	0.71×10^{-2}	3.68	0.099947
2-BEST	0.79	0.06	4.13	0.091043
4-BEST	1.10	0.20	4.95	0.085659
16-BEST	2.90	0.21	5.88	0.083308

Table 2 Runtime in seconds of non-ML MIMOPack detectors for very large MIMO systems

$n_T = n_R$	ZF-SIC		2-BEST	
	Sequential	32 threads	Sequential	32 threads
20	0.11	0.02	2.28	1.16
40	0.30	0.03	6.40	0.44
60	0.71	0.07	13.04	0.94
80	1.22	0.11	21.59	2.29
100	1.78	0.15	32.45	2.44

signals on the sequential CPU (with one OpenMP thread) and the time to execute the same simulation on a multicore and GPU system.

As we can see in Table 1, we have a good speedup for the multicore version for all kind of detectors. MLE GPU version exhibits a good performance even better than that obtained with multicore version. However, due to the low complexity and the non-parallel pattern of the suboptimal methods, ZF-SIC and K-BEST methods have a higher execution time than sequential version because the time needed to transfer data between the CPU and the GPU is greater to the detection time itself. Therefore, it is necessary to increase the computational burden of the simulation to obtain good performance with GPUs.

Table 2 shows the execution time of the non-ML detectors for Massive MIMO systems. The simulation have been done with the same parameters of the Table 1 but increasing from 20×20 to 100×100 the MIMO system size.

5 Conclusions

This paper presents a high-performance library for MIMO communications systems which aims to provide a set of routines needed to perform the most complex stages in the current wireless communications. The proposed library exhibits three important features: portable, efficient and user friendly. These aspects make this library a very useful tool for companies involved in the development of new wireless and broadband standards, which need to obtain results and statistics of its proposals quickly and also for other researchers making easier the implementation of scientific codes. The use of the library in heterogeneous mode allows the user to simulate the behavior of the system with multiple values, thereby providing greater versatility and speedup. The efficiency of MIMOPack has been evaluated by comparing the BER and computational time with different types of hard-output detectors and platform configurations. The variety of detectors with mixed complexities and performances allows to cover multiple use cases with different channel conditions and scenarios such as massive MIMO. Moreover, parallel implementations allow the execution of large simulations over different architectures thus exploiting the capacity of the modern machines.

Acknowledgments This work has been supported by SP20120646 project of Universitat Politècnica de València, by ISIC/2012/006 and PROMETEO FASE II 2014/003 projects of Generalitat Valenciana; and has been supported by European Union ERDF and Spanish Government through TEC2012-38142-C04-01.

References

1. Paulraj AJ, Gore DA, Nabar RU, Blcskei H (2004) An overview of MIMO communications—a key to gigabit wireless. Proc IEEE 92(2):198–218
2. Rusek F, Persson D, Lau B, Larsson E, Marzetta T, Edfors O, Tufvesso F (2013) Scaling up MIMO: opportunities and challenges with very large arrays. IEEE Signal Process Mag 30(1):40–60
3. Lin Y, Lee H, Woh M, Harel Y, Mahlke S, Mudge T, Chakrabarti C, Flautner K (2007) SODA: a high-performance DSP architecture for software-defined radio. IEEE MICRO 27(1):114–123
4. Yang C-H, Markovic D (2008) A multi-core sphere decoder VLSI architecture for MIMO communications. Global telecommunications conference, pp 1–6
5. Wu D, Eilert J, Liu D (2011) Implementation of a high-speed MIMO soft-output symbol detector for software defined radio. J Signal Process Syst 63(1):27–37
6. Tan K, Liu H, Zhang J, Zhang Y, Fang J, Voelke GM (2011) Sora: high-performance software radio using general-purpose multi-core processors. Commun ACM 54(1):99–107
7. Wu M, Sun Y, Gupta S, Cavallaro J (2011) Implementation of a high throughput soft MIMO detector on GPU. J Signal Process Syst 64(2):123–136
8. Nylanden T, Janhunen J, Silven O, Juntti M (2010) A GPU implementation for two MIMO-OFDM detectors. International conference on embedded computer systems, pp 293–300
9. Falcao G, Silva V, Sousa L (2009) How GPUs can outperform ASICs for fast LDPC decoding. International conference of supercomputing, pp 123–136
10. Innovative Computing Laboratory, University Tennessee, Knoxville (2009) MAGMA: Matrix algebra on GPU and multicore architectures. Available at http://icl.cs.utk.edu/magma/index.html
11. EM Photonics, Inc (2010) CULA Tools - GPU accelerated LAPACK. Available at http://www.culatools.com
12. MathWorks, Inc. (2011) Communications System Toolbox - Design and simulate the physical layer of communication systems. http://www.mathworks.es/products/communications/
13. ITPP-C++ Library for Mathematical, signal processing, speech processing, and communications classes and functions. Available at http://itpp.sourceforge.net
14. Roger S, Ramiro C, Gonzalez A, Almenar V, Vidal AM (2012) An efficient GPU implementation of fixed-complexity sphere decoders for MIMO wireless systems. Integr Comput-Aided Eng 19(4):341–350
15. Ramiro C, Roger S, Gonzalez A, Almenar V, Vidal AM (2013) Multi-core implementation of a fixed-complexity tree-search detector for MIMO communications. J Supercomput 65(3):1010–1019
16. Garcia VM, Gonzalez A, Gonzalez C, Martinez-Zaldivar FJ, Ramiro C, Roger S, Vidal AM (2011) The impact of GPU/multicore in signal processing: a quantitative approach. Waves 3:96–106
17. Roger S, Ramiro C, Gonzalez A, Almenar V, Vidal AM (2012) Fully parallel GPU implementation of a fixed-complexity soft-output MIMO detector. IEEE Trans Veh Technol 61(8):3796–3800

18. Domene F, Roger S, Ramiro C, Piero G, Gonzalez A (2012) A reconfigurable GPU implementation for Tomlinson–Harashima precoding. 37th international conference on acoustics, Kyoto, Japan
19. Domene F, Roger S, Ramiro C, Piero G, Gonzalez A (2012) Efficient implementation of multiuser precoding algorithms on GPU for MIMO-OFDM systems. XXVII Simposium Nacional de la Unin Cientfica Internacional de Radio, Elche, Spain
20. Ramiro C, Simarro Haro MA, Martinez-Zaldivar MJ, Vidal AM, Gonzalez A (2013) A GPU implementation of an iterative receiver for energy saving MIMO ID-BICM systems. J Supercomput. doi:10.1007/s11227-013-1081-x
21. Larsson EG (2009) MIMO detection methods: how they work [lecture notes]. Signal Process Mag IEEE 26(3):91–95. doi:10.1109/MSP.2009.932126