

IMPLEMENTATION OF 802.11n ON 128-CORE PROCESSOR

A. Akapyev
Nizhny Novgorod State Technical University
Nizhny Novgorod,
603950, Russia
inventor@inventor.nnov.ru

V. Krylov
Mera Labs LLC
Nizhny Novgorod,
603163, Russia
vkrylov@meralabs.com

Abstract

This article presents the results of a research in applying modern Graphics Processing Units in the field of telecommunications. The most recent Wireless Local Area Network protocol, 802.11n, was studied, as it introduces a significant increase of computational complexity. Taking into consideration the concept of Software Defined Radio, the implementation of PHY algorithms was devised on a modern programmable 128-core GPU. It was shown that a significant performance increase could be obtained through applying this novel approach. Our experiments had shown more than 8x performance boost.

The article also discusses the approach of utilizing a GPU as a networking device. The author shows how a GPU could be effectively used as a Digital Signal Processing unit of a real communication device.

1. INTRODUCTION

Graphics Processing Units attract the attention of non-graphics persons for a number of years already. There are many areas where fast computations are a must. At the initial stages of applying a GPU as a general purpose computing device [3] there were several significant programming limitations:

- the only available graphics programming APIs narrowed down the range of algorithms that could be effectively implemented on GPUs due to the absence of the ‘scatter’ operation and fixed pipeline;
- a low precision of storage and computing on floating point numbers;
- integer and logical arithmetic was not available.

Under a strong user demand two major hardware vendors released non-graphics programming interface specifications for their GPUs. The first was AMD/ATI, which introduced the CTM (Close To Metal) [4] programming environment in mid-2006. The CTM provided an assembly-level programming interface revealing the most of the hardware architecture. In

the end of 2006 NVidia followed with a G80 chip and CUDA environment [5]. CUDA provided a consistent C-based programming environment.

To exploit the enormous power of modern programmable GPUs in signal processing tasks, the author chose CUDA as the easiest one to learn and use.

2. CUDA COMPUTING MODEL

The CUDA computing architecture can be seen as a shared-memory system. The programming model encourages a SIMD style when the user writes a function called *kernel* which is executed by all the available processors at the given moment. At the highest level the device is represented by a *grid* composed of blocks, where each block contains a number of threads, figure 1.

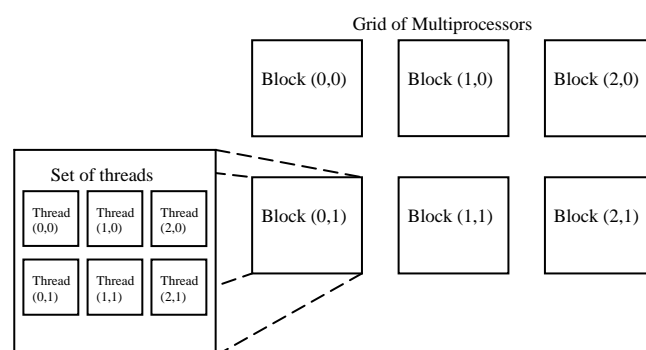


Figure 1. CUDA computing model.

A grid has a two dimensional structure and a block has a three dimensional one (figure 1 illustrates that we are using only two dimensional addresses).

3. 802.11N CORE ALGORITHMS

The 802.11, a widespread Wireless Local Area Network technology, had undergone many modifications to increase the network throughput. The recent 802.11 specification [1] had already incorporated the FHSS, DSSS and OFDM PHY technologies. The newer technologies pose higher demands on the signal processing hardware. The upcoming 802.11n [2] addition introduces the MIMO-OFDM technology which is likely to require more than twice of processing power than any previous PHY technology did.

3.1. MIMO-OFDM

The Orthogonal Frequency Division Multiplexing in combination with the Multiple Input – Multiple Output technology exploits frequency selectivity and spatial diversity on the wireless channel. The block scheme of the MIMO can be seen on figure 2.

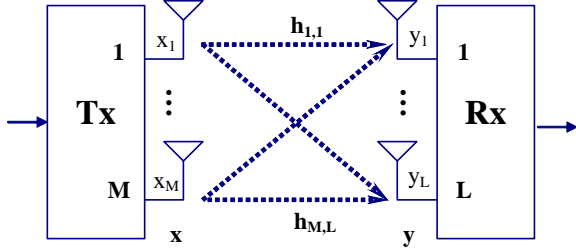


Figure 2. MIMO block-scheme.

The modulated symbols' stream flows through the serial-to-parallel convertor which divides it into M parallel streams, where M denotes the number of transmitting antennas. All M antennas simultaneously transmit M different symbol streams. Each of L receiving antennas detect all the M transmitted streams. Due to the properties of the wireless channel, transfer coefficients from each transmitting antenna to each receiving antenna are all different and denoted as h_{ij} , where i stands for the order number of a transmitting antenna and j stands for the order number of a receiving antenna. The set of transfer coefficients $\{h_{ij}\}, i = \overline{1, L}; j = \overline{1, M}$, form the MIMO channel matrix H. Therefore the received signal on the end of the MIMO on the interval of a single OFDM symbol is given by the following formula:

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{n}, \quad (1)$$

where y is the received signals' vector; x is the transmitted signals vector; H is the MIMO channel matrix; n is the vector of white Gaussian noise realizations.

Assuming that transmitter has the knowledge of the MIMO channel matrix H, the received signal y could be obtained by applying the Maximum Likelihood Algorithm (MLA). According to the MLA, the received signal estimate is given by the formula:

$$\hat{\mathbf{x}} = \arg \min_x \|\mathbf{H}\mathbf{x} - \mathbf{y}\|, \quad (2)$$

where $\|\cdot\|$ denotes the norm operation.

According to the formula (2), it is needed to calculate the decision function for every possible vector

x and take as an estimate the vector x for which the decision function is minimal. The algorithm's complexity is $O(M^N)$, where M is the number of transmitting antennas and N is the number of bits per symbol.

4. IMPLEMENTATION OF MIMO ON 128-CORE PROCESSOR

For the implementation, a QAM16 modulation scheme and 2x2 MIMO system were chosen.

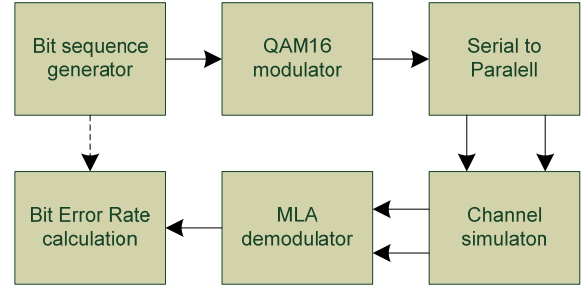


Figure 3. Simulator block-scheme.

Figure 3 shows the implemented MIMO simulator structure. All the blocks shown on the figure were completely implemented in the CUDA and executed on a GPU. The QAM16 modulator is the simplest block as it is a constellation mapper which maps every 4-bit sequence to the corresponding I and Q numbers. Here the serial-to-parallel block is a virtual block since the CUDA has a shared-memory architecture and the channel simulation block simply takes data from particular memory addresses. The channel simulation multiplies complex samples by the propagation coefficients represented by the channel matrix H.

The MLA Demodulator is the most complex block as it reduces all the possible transmitted vectors to a single estimate. For the 2 transmitting and 2 receiving antenna configuration we can describe decision process as the calculation of the decision function (2) in each entry of the matrix. A column number of such a matrix designates a possibly transmitted symbol on the first transmitting antenna and a row number designates a possibly transmitted symbol on the second one. The minimum of the decision function (2) will point to the estimate. At the beginning of the algorithm's work the first group of 16 threads in a block scan for the minimum decision row-wise, each thread scans a particular row; the other 16 threads do the same column-wise. In the same way the algorithm reduces the decision to a single entry.

The Bit Error Rate calculation is performed via calculating differences between transmitted and re-

ceived bit sequences fully in parallel and then performing parallel sum reduction.

5. PERFORMANCE CHARACTERISTICS

The parallel simulator implemented in Section 4 was compared with the sequential implementation. The NVIDIA 8800 GTX GPU was used.

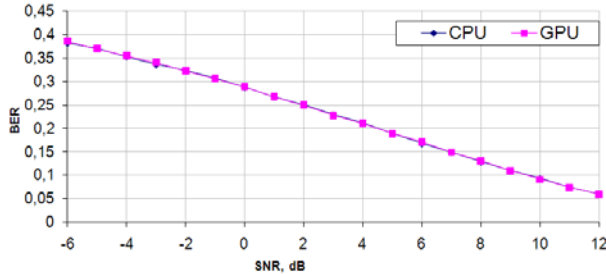


Figure 4. CPU and GPU simulation BER(SNR).

From Figure 4 we can see that the curves of the resistance to interference are identical in both cases, from what we can conclude that the calculation precision on the GPU is sufficient for such a kind of simulations.

Table 1. CPU vs. GPU performance comparison

	CPU – simulator	GPU – simulator
Hardware model	Intel E6600	NVIDIA 8800 GTX
Execution time of simulation transferring 1000 MIMO-symbols, 192-bits each.	14.422 s.	1.585 s.
Execution time of simulation transferring 1000 MIMO-symbols, 256-bits each.	18.453 s.	2.132 s.
Execution time of simulation transferring 1000 MIMO-symbols, 512-bits each.	37.266 s.	4.184 s.

The performance results given in Table 1 show that the NVIDIA 8800 GTX GPU outperforms the Intel

6600 CPU 8 to 9 times. These are great results and further GPU code optimizations can give even significantly better ones [6].

6. GPU AS A NETWORKING DEVICE

According to the Software Defined Radio (SDR) approach [7], the implemented simulator could be used as a signal processing unit of a Software Defined Radio. A simplified SDR system model is shown on Figure 5.

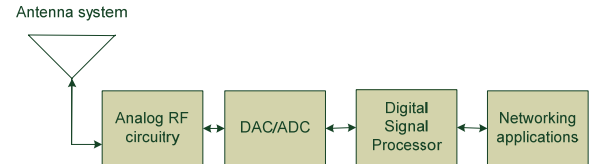


Figure 5. Simplified Software Defined Radio system model.

The antennas, analog RF circuits and DAC/ADC are all only the hardware part of the system. They are relatively versatile devices and could be used to build very different systems [9]. The specifics of protocols and applications are concealed in the two rightmost blocks shown on Figure 5.

It has already been shown how the MIMO PHY layer could be implemented on the commodity GPU serving as a Digital Signal Processor in the previous sections of the article. Now we'll place emphasis on how to connect the networking applications running under the PC operation system with the graphics board serving as a DSP.

6.1. Network Interface Driver

To enable the use of the Graphics Processor Unit as a Digital Signal Processing unit for 802.11n, a Windows network interface driver was developed. This driver represents a regular network interface such as the Wi-Fi or Ethernet card have in their system.

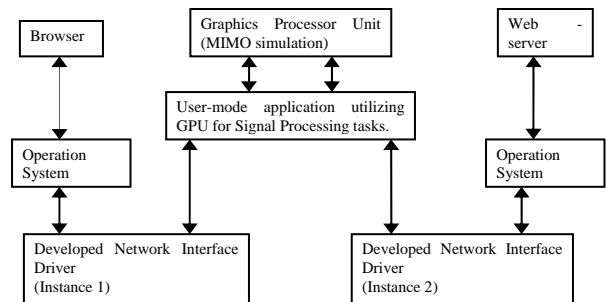


Figure 6. Employing GPU for Digital Signal Processing.

Figure 6 shows a particular use case where a developed network interface driver connects a Browser network application to a GPU performing a MIMO simulation. The principal difficulty is that CUDA could be used only by a user-mode application and the network driver could only be run from the kernel-mode. For this purpose, a specific user-mode application the driver supplies network packets to was created. On Figure 6 we showed that several networking applications could be connected to the same user-mode application which controls the GPU. In such a way we built a robust network packet transmission simulation via the PHY layer of 802.11n. The simulation could be run entirely on one PC with the NVIDIA G80 graphics board.

It is necessary to note that the instances 1 and 2 of developed network drivers (figure 6) should belong to different networks in terms of the Internet Protocol. For packets to flow through the system, it is needed to change IP addresses in packets not to confuse the OS IP stack, Figure 7.

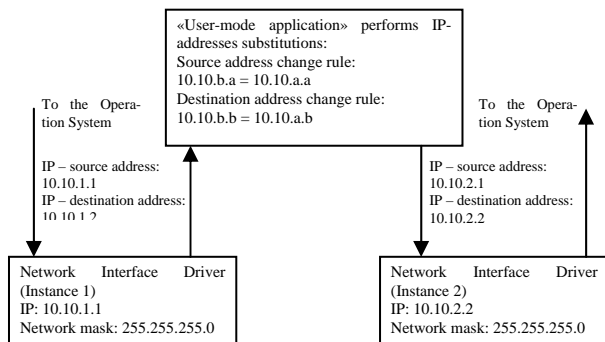


Figure 7. Addresses substitutions.

The corresponding substitution of MAC addresses is also necessary.

6.2. Simulation results

The implemented simulator was successfully used in the dependency evaluation of physical level performance simulated on the GPU concerning network applications' performance. We can successfully transfer file-stream video over the simulator with the structure given on Figure 6.

By varying the signal-to-noise ratio on the PHY layer, we have experimentally determined that, with the SNR higher than 15 dB, a user could watch video without any interference, but with the SNR lower than 8 dB, video streaming is not feasible in the given system. We should note that the Forward Error Correction was not applied at that moment.

7. CONCLUSION AND FUTURE WORK

This article presents an approach to the implementation of Software Defined Radio units on a Graphics Processor Unit. As far as we're aware, there are no other papers on this topic. It was shown that modern GPUs could be successfully employed as powerful Digital Signal Processors running wireless algorithms. It was demonstrated that a MIMO 2x2 simulation on a GPU is eight to nine times faster than on a CPU. Further optimizations may significantly increase this performance gap. It was demonstrated how the PHY layer performance could be evaluated on the application layer, respective to the OSI model. The future work will include the implementation of a convolution coder/decoder and the emphasis will be placed on the efficiency of implementations.

8. REFERENCES

- [1] IEEE 802.11-2007. IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 2007.
- [2] IEEE 802.11n/D2.00. Draft STANDARD for Information Technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Amendment: Enhancements for Higher Throughput, 2007.
- [3] General Purpose Computation Using Graphics Hardware, <http://www.gpgpu.org>.
- [4] AMD CTM, <http://ati.amd.com/companyinfo/researcher/documents.html>.
- [5] NVidia CUDA, <http://developer.NVidia.com/object/CUDA.html>.
- [6] Shane Ryoo, Christopher I. Rodrigues, Sara S. Baghsorkhi, Sam S. Stone, David B. Kirk, Wen-mei W. Hwu. "Optimization Principles and Application Performance Evaluation of a Multithreaded GPU Using CUDA", Proc. ACM SIGPLAN, pp. 73-82, 2008.
- [7] Walter Tuttlebee. Software Defined Radio. Origins, Drivers and International Perspectives. John Wiley & Sons Ltd., Chichester, 2002.

[8] Henrik Schulze, Christian Luders. Theory and Applications of OFDM and CDMA. John Wiley & Sons Ltd., Chichester, 2005.

[9] Universal Software Radio Peripheral,
<http://www.ettus.com>