Applying Parallel and Distributed Computing Curriculum to Cyber Security Courses

Radu Velea, Valentin Ilie, Ion Bica
Faculty of Information Systems and Cyber Security
Military Technical Academy
Bucharest, Romania
radu.velea@mta.ro, vilie@fitbit.com, ion.bica@mta.ro

***Abstract*—Parallel technologies evolve at a fast rate, with new hardware and programming frameworks being introduced every few years. Keeping a Parallel and Distributed Computing (PDC) lecture up to date is a challenge in itself, let alone when one has to consider the synergies between other courses and the shifts in direction that are industry-driven and echo inside the student body. This paper details the process of aligning parallel and distributed curriculum at the Military Technical Academy of Bucharest (MTA) over the last five years, with government and industry demands as well as faculty and student expectations. The result has been an adaptation and an update of the previous lectures and assignments on PDC, and the creation of a new course that relies heavily on parallel technologies to provide a modern outlook on software security and the tools used to combat cyber threats. Concepts and assignments originally designed for a PDC course have molded perfectly into a new supporting paradigm focused on malicious code (malware) analysis.**

***Index Terms*—Parallel Programming, Malware Analysis, GPU Computing, Undergraduate Education**

I. Introduction

From smartphones to supercomputers, most hardware devices today implement some form of parallelism. Multi-core, superscalar processors have made the world we live in more interconnected than ever-before. Information travels between user-terminals and the cloud, and creates virtual systems that can fulfill the everyday needs of corporations and individuals alike. The hardware and software systems that make up our everyday life posses a level of complexity that is not immediately perceivable by people that are not educated in the matter. The engineers that make and keep these systems running have to do so efficiently and safely. Industry organizations and academia must constantly adapt as technology evolves, and take into consideration the realities present in *the field* when selecting and educating their rank and file. Our goal is to provide quality education that allows future graduates to tackle complex problems in a broad range of fields.

Recent years have seen an increase in demand for cybersecurity specialists. To meet these demands local universities have started to develop new courses. As performance and security drive almost every modern engineering endeavor, universities have found themselves racing to incorporate as many new concepts as possible from the fields of high performance computing and cybersecurity into their curriculum [1]. While creating the curriculum for a new course focused on malware analysis we realized that most real-life problems in the field require prior knowledge of PDC concepts that students learn later in their academic career. To solve this problem we decided to introduce PDC curriculum earlier in the academic cycle to facilitate the understanding of how modern anti-malware engines work and the nature of the threats they are protecting us from. By exposing students to more PDC content we also hoped to give them the tools to work on projects that better reflect the scenarios they will encounter in practice.

The current paper presents our two-pronged effort at the Military Technical Academy[1] of Bucharest to update PDC curriculum and, later, use elements of the PDC lecture to augment other courses that cover aspects such as advanced computer architecture and software security. The theoretical content is derived from older courses from MTA or the curriculum from the equivalent universities in Romania, especially Politehnica University of Bucharest (UPB [2]). Novel elements introduced in the PDC lecture have closely followed recent hardware advances, with a focus on open standards and availability. The emphasis of this interdisciplinary mix is how PDC content can empower students to understand and fight against malicious code and other cybersecurity threats.

II. Motivation and Context

Engineering and computing courses from most universities often introduce their students to sequential algorithms or programming techniques. Looking at the world in a parallel and distributed paradigm comes at a later educational stage and might be perceived as an advanced or niche skill. The prerequisites for understanding all the underlying concepts work to trim the target audience and lead to suboptimal assimilation of PDC courses [3]. Subpar assimilation of PDC concepts has the tendency to snowball into other areas of computer science and undermines early academic and industry efforts to make sure future engineers reach their full potential. The motivation for the effort described below is to efficiently use existing resources and expertise to develop quality content for the students at the Military Technical Academy of Bucharest.

A. Institution Description

Military Technical Academy of Bucharest is a polytechnic university which trains engineers for the Ministry of National

[1]Military Technical Academy of Bucharest - https://www.mta.ro/

Defense and other structures of the national defense system. It is under the coordination of the Ministry of National Defence and offers technical education up to PhD level. Its student body is composed of military and civilians alike. It is the country's foremost institution for training officers in the field of engineering. Graduate students from the MTA occupy various roles in the national defense organizations or private industry.

The object of this paper are the software engineering courses from the Faculty of Information Systems and Cyber Security. The faculty runs one of the oldest and well recognized master program in cyber security in Romania and, four years ago, it started an undergraduate program in cyber security besides the existing one in computer engineering. The faculty enrolls between 60 and 100 undergraduate students yearly (the number may vary according to state and industry demands). Students are organized into groups of 20.

B. Motivation for Parallel Courses

Students follow a 4-year, conventional, computer engineering curriculum which consists of undergraduate courses in mathematics, programming (data structures, algorithms, object oriented programming, etc.), electronics, networking, databases, operating systems, cryptography, Web technologies, and PDC. The Parallel Architecture (PA) lecture is presented in the 7th semester when students already have a solid understanding of programming, networking and operating systems. Before the first academic year of the timeline described in this paper, faculty management decided to update curriculum and add new contents to provide graduate engineers enhanced skills in solving parallel and distributed problems.

C. Motivation for Security Courses

Those with the computing power and corresponding speed of information flow have a tremendous advantage on the battlefield. Throughout history, whoever analyzed and acted faster, won. With this statement, Gen. Ronald R. Fogleman, Air Force chief of staff, started to define Information Operations as a 5th dimension alongside land, sea, air and space [4]. Recent technological advancements and the geopolitical evolution have determined the North Atlantic Treaty Organization (NATO) to integrate sovereign cyber effects, provided voluntarily by Allies, into Alliance operations and missions [5], making cyberwar a core technical component of the 5th dimension [6]. This represents a motivation for universities to invest more into courses and programs that put an emphasis on cybersecurity. This is especially true for an institution like MTA, that is expected to provide engineers for organizations responsible with regional and global cyber defense.

D. Timeline

The evolution our courses was implemented in several stages. In the first stage we resurrected the Parallel Architecture course and, over the course of 2 years, we experimented with technologies the students had not been exposed before: instruction level SIMD programming (SSE and AVX extensions from Intel®, OS-agnostic thread API (C++11) and GPU programming (OpenCL).

After gathering feedback, we moved some of the low-level architecture content and assignments to the 2nd year lecture on Computer Architectures and replaced it with more content on parallel computing. In the previous year, the faculty had created a new specialization with focus on cybersecurity and the students enrolled there were among the first to experience this content at such an early stage. In the 3rd stage two new courses were added for the new cybersecurity specialization: a spin-off of the existing Operating Systems lecture with more focus on security and Malicious Code Analysis (MCA). In the 4th stage, after 2 years of experiments and improvements, the MCA lecture would become available for all the students.

III. Parallel Architectures Course

This section describes the structure of the Parallel Architecture course at the MTA and presents the student feedback and lessons learnt that allowed us to move forward and use PDC content to improve other lectures.

A. Related Works

We used [7], [8] and [9] for theoretical reading material. There were, also, a multitude of available online resources in terms of workshops, lectures or other form of documentation that we don't have the space to enumerate here in full. We tried to make the students familiar with as many introductory concepts as possible and then focus on practical activities.

B. Theoretical Course Structure

Twelve out of fourteen weeks of the academic semester reserved with presenting PDC topics; the duration of each lecture was 2 hours. The last two were dedicated to individual assignments where each student had the freedom to select a subject related to PDC and make a 15-minute presentation in front of the class. The main topics covered were presented in the following order:

1) Applications of data parallelism
2) Parallel algorithm complexity
3) Synchronization and inter process communication
4) Parallel search algorithms
5) OpenMP
6) Distributed computing with message passing interface (MPI)
7) Hybrid parallel implementations
8) Wave and leader election algorithms
9) Vector instructions: SSE, AVX
10) GPU programming with OpenCL
11) Image processing with parallel technologies

The first lecture was spent on introduction and familiarization with debugging and performance analysis tools. Students familiar with sequential programming might have trouble debugging multithreaded applications and could struggle when encountering race conditions, deadlocks or nondeterministic behaviour. From the onset we put an emphasis on performance

and how to measure it, as one of the main reasons for using more computational resources and parallelism to solve a problem is performance. We defined performance in terms of speed and efficiency (how much power a system uses to solve a problem, the total amount of memory, network traffic, etc.) and challenged the students to identify trade-offs between different implementations.

In the next lectures we jumped into Flynn's taxonomy [10], covered system APIs (POSIX and WinAPI) and C++11 support for working with threads. The programming language of the course was C/C++. There were some references to python and Java, but all case studies and examples were written in C/C++. At this point the students were ready to experiment with various problems and we noticed an increased enthusiasm when the case studies presented or their assignments performed better in terms of performance compared to sequential versions.

After we were confident that the students had a good understanding of the previous topics we introduced them to OpenMP and MPI. After another week we presented hybrid implementations that used both frameworks. We will later use this as an analogy when discussing OpenCL's programming model or hybrid CPU-GPU implementations.

The inclusion of Streaming SIMD Extensions (SSE) and OpenCL in the curriculum was a pragmatic decision. We thought SSE would appeal to the students because of its relative novelty compared to 16-bit Intel®80386 that was studied in the 2nd year's Architecture course. We did not have sufficient resources in terms of hardware to support GPU programming on Nvidia in our laboratories. We made the decision to allow each student to use his or her laptop instead and took the calculated risk that none of those devices would be more than 15 years old (SSE2 is available for all x86 CPU generations since Pentium 4, 2001). Intel and AMD OpenCL SDKs can be used on machines that have integrated graphics or provide CPU-only compatible devices. In other words, we found that SSE2 and OpenCL were the lowest common denominator that allowed us to exemplify the theoretical concepts included in the course. Programming with SIMD instructions paved the way for a better understanding of OpenCL's vector operators and data types. The previous MPI/OpenMP case studies had prepared the students for the host-device paradigm specific to GPU programming.

As an epilogue to the course content we presented how all these concepts and technologies can be used to speedup performance-critical applications. We focused problems requiring image processing.

C. Practical Activities and Assignments

The laboratories were oriented towards programming and performance analysis and lasted for 2 hours. Students had to use C/C++ to optimize a range of problems: matrix multiplications, signal processing, etc. Their work over the course of a lab session was graded and made up, in total for 10% of their final score. The percentage was chosen to encourage engagement, but also to maintain a relaxed atmosphere by not significantly impacting their evaluation. A few selected example of practical lab activities are listed below:

- Multithreading programming with Posix, WinAPI and C++11 threads
- Synchronization problems with threads - one lab was dedicated to a problem involving network traffic interception with libpcap[2] and parallel processing the contents
- Acceleration of arithmetic operations with OpenMP
- Parallel sorting [11] and compression - embarrassingly parallel tasks
- MPI examples
- Vector and matrix operations (multiplication, decomposition, inversion) with SSE
- Matrix and image processing with OpenCL

Homeworks were structured into 3 assignments. Completion of any two assignments equivalates to 30% of the final grade. Being a new course, we deliberately added an extra assignment to see how students would react to the content and what technologies they prefer, being given the choice. The assignments themselves conferred a significant level of freedom with regard to the technology used.

The first homework consisted in creating a library for parallel processing PPM images[3]. Students were required to create functions for applying different filters to input images. The library integrator could choose the number of threads that the library would use when performing the tasks. One extra task consisted in a *Where's Waldo?*[4] challenge where students had to compete with each other to find objects in an image. For this sub-task, the students were required to select the optimum number of threads. Bonus points were given for speed and accuracy.

The second home assignment was another library for parallelizing mathematical problems. It ranged from simple integer vector computations to floating point matrix multiplications, inversions and decompositions. Students were required to use OpenMP. Bonus points were awarded for the fastest speedups.

The last assignment was a parallel texture compressor. Students could use any technology studied during the course (C/C++, OpenMP, SSE/AVX, OpenCL) to build a DXT1 [12] compressor for PPM images. Students were graded based on a combination of the quality and speedup of their implementation. They were required to meet certain quality thresholds for each image in the testbed and a minimum speedup. Quality was measured as peak signal-to-noise ratio (PSNR). All homework was submitted on the open source automated testing platform Vmchecker [13] and tested against a naive, serial implementation of the algorithm.

D. Results and Student Evaluation

The final examination consisted of a written test that combined theoretical and practical elements. In the first year there were 57 students enrolled in the course (3 having

[2]tcpdump & libpcap - https://www.tcpdump.org/

[3]Netpbm color image format - http://netpbm.sourceforge.net/doc/ppm.html

[4]Where's Wally? - https://en.wikipedia.org/wiki/Where%27s_Wally%3F

dropped out the previous years), 17 scored maximum points and only 2 failed the final examination. The second year, out of 47 remaining students (out of the original 60), 10 scored maximum points and 2 failed.

From the presentation topics that the students chose for the final weeks two stand out: the passion for Bitcoin mining on the GPU and the impact of parallel technologies on security applications. Some students moved on to study advanced PDC elements in graduate schools and some chose to write their bachelor thesis or dissertation on a related topic. Some of their works have been published [14].

E. Student Feedback

Throughout the course we experimented with active learning strategies and gathered impressions from students as they encountered course topics [15]. At the end of the year, after the final examination, the students were asked to fill an anonymous feedback form. The feedback, as shown in Fig. 1, was overwhelmingly positive.

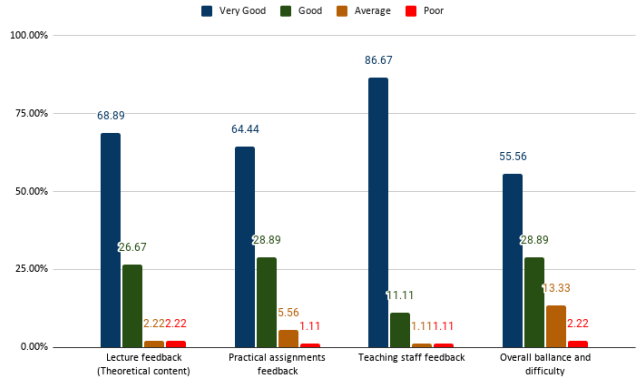

Fig. 1. Student feedback after changes made for Parallel Architecture course.

The students were asked to express how they felt about the course contents: theoretical and practical aspects, professors, assistants, and difficulty of the final examination. Each category received had to be marked as one of *Very Good*, *Good*, *Average (Neutral)* or *Poor*. They were also asked to highlight the best aspects currently present and something that can be improved in the future.

The most common grievance among students was that the lecture contained lots of things that were totally new to them and they sometimes felt that more time should have been spent on a certain subject. Some students wrote that they would like to experiment with CUDA and that parts of the knowledge base presented in the course would have been equally interesting to them in their first years at the university. To address this issue we decided to expose students to certain PDC topics earlier in the undergraduate cycle by introducing them in the Computer Architectures, Operating Systems and, later, Malicious Code Analysis curriculum.

Contrastingly, exposure to novel technologies was often mentioned as one of the positive aspects of the course in the feedback form. The students liked to see the examples that were presented in the lab or their own code evolve. It become faster as the semester progressed and they became more accustomed to optimization techniques and parallel algorithms. The fact that assignments were competitive in nature and had some gamification elements was also very well received. Students had to compete against each other for the best implementation, they worked individually or in teams (depending on the assignment type) and had their scores published weekly on leaderboards that contained their best performances and total points obtained during the semester. Encouraged by this experience and some recent studies on the subject [16] we decided to further continue including these types of exercises in the new security course.

IV. Malicious Code Analysis Course

To align with the needs of some government institutions and local companies, MTA education management decided to create new courses focused on combating modern security threats. For real-time detection and prevention techniques to be efficient in a modern computing environment (highly-interconnected, with lots of data traveling between terminals and lots of events firing asynchronously [17]) they had to make use of some sort of parallelism and distributed technology [18].

A. Synergies with Existing Lectures

Student response to PDC elements introduced in the Parallel Architectures course suggested that some of them could be taught earlier in the academic cycle. Designing modern security solutions requires intimate knowledge of CPU and operating system architecture. Processing events and data in search of incidents or threats demands highly parallel software [19], with hundreds of threads, interaction with the cloud [20] and sometimes the use of a GPU [21]. It was clear that in order to build a new course on Malicious Code Analysis we needed to integrate elements from PDC in the existing curriculum.

B. Moving Content from Parallel Architectures

We identified Computer Architecture (CA) and Operating Systems (OS) courses as two prerequisites for the new cybersecurity course. We moved some elements from the 4th year Parallel Architectures to the 2nd year Computer Architecture course:

- vector (SIMD) instructions
- multi-core architectures
- GPU architecture and memory model
- using the above to solve embarrassingly parallel problems

We created a gradual transition from 8086 (16-bit registers) to Skylake[5] (64-bit registers). The parallel pipeline of the modern processor and the multi-core architecture was introduced in the lecture. In the PA course SSE/AVX programming was done using intrinsics[6], but we decided that students would benefit more if they used assembly instructions directly. The increased

[5]Intel® 6th-generation Core
[6]https://software.intel.com/sites/landingpage/IntrinsicsGuide/

focus on assembly language programming and debugging would be very useful in security-related tasks down the line. We also noticed that assembly programming became more attractive to students as the registers increased in size.

The cybersecurity student group that first reached its 3rd year was introduced, in the first semester, to a new course called Operating System Security (SSO). The course exposed students to some advanced security content and included lectures with material from the Parallel Architectures course:

- processes and scheduling algorithms
- inter-process communication
- asynchronous I/O
- threads and SIMD
- performance analysis

PDC contents such event-driven programming was presented in the context of software security. This contributed to the creation of a knowledge base that could be expanded in the upcoming semesters.

C. Course Structure

The first instance of the Malicious Analysis Course debuted with a group of 20 students that had opted for the cyber security specialization two years prior. The major objectives of the course were to familiarize students with the best practices of writing secure C/C++ code, reverse engineering and malware analysis. Students would be required to read C/C++ and assembly language code and identify faulty or malicious behaviour. The lecture would walk them through the inner works of modern protection tools: Network Intrusion Detection and Prevention Systems (NIDPS) such as Snort[7] and Suricata[8] or host-based projects such as ClamAV[9] - all of them capable of using SIMD parallelism for malware detection. The main course topics are listed below:

1) Architecture, compiler and operating systems overview
2) Importance of assembly language and low-level programming in security
3) Payloads and calling conventions
4) Executable file formats: PE, ELF, Mach-O
5) Loading and running executable files from the OS point of view
6) Reverse engineering
7) Sandboxing and emulation
8) Signature-based detections
9) Network threats and defenses
10) Types of malware
11) Modern threats and recent security incidents
12) Case studies

The course is concerned with code analysis that runs on Windows and Linux operating systems. The students are required to use system analysis tools to gather information about the execution environment and file system. Assignments require x86 assembly language, C/C++, and a bit of Python. Code analysis is done statically and dynamically. For dynamic analysis students have to write programs to monitor systems (list and read active processes) and learn to analyze a live process (extracting instruction flows or data sections from it). Encryption, binary compression and the methods to deal with them in security software are discussed. Static code analysis consists of searching for known strings inside a file (signature-based detections). Both static and dynamic analysis can be enhanced by training machine learning models to identify certain patterns or to detect anomalies (clustering algorithms).

D. Using Parallel and Distributed Computing to Improve Software Security

PDC topics were required for case studies and practical applications. Each week's presentation was accompanied with a case study that could be continued as homework. For example SSE/AVX programming was used for accelerating pattern-matching code, encryption algorithms and for showcasing evasion mechanisms used by malware such as Emotet[10]. Distributed computing was useful in understanding botnets and their command-and-control system.

A recurrent assignment was for students to write a daemon or service to monitor activities inside the operating system. The daemon would attempt to extract information from the system files, running processes and network interface, and process it in real time. The processing could be done on the same machine or on a server. The challenges here were to extract useful information from system with the least amount of privileges required and to efficiently process and interpret the extracted data. Extracting information from live processes or the network interface would require the use of IPC. Information processing would consist of string matching, clustering (for anomaly detection) or other heuristics. Given the amount of data that can be extracted, even from a simple personal computer, this process would greatly benefit from parallelization. Basically, students were taught to use a parallel and distributed architecture to make their security applications scale in a realtime scenario. Some of the PDC concepts used in the cybersecurity course had been integrated in previous lectures, while others such as SIMD programming, distributed computing and shared memory were new to the students. The new topics were learnt on-the-fly as ways to solve modern security problems. The general idea was to enable students to identify and analyze malicious code, and design software that provides efficient realtime detection.

E. Informal Learning

The duration of the course and the number of lectures in a semester could not hope to provide in-depth cover for all aspects related to malicious code analysis. Building on the lessons learned from PA, to expand the scope of the course we introduced an element of informal learning that awarded extra credits (up to 20%) to students. Students were encouraged to select a topic related to the course and create a project and

[7]https://www.snort.org
[8]https://suricata-ids.org/
[9]https://www.clamav.net/
[10]https://www.malwarebytes.com/emotet/

presentation about it (similar to PA). Students were given some topic examples, but were otherwise encouraged to pick something on their own. Unlike PA, where presentations would be individual, the students were allowed to group in teams of two for their projects. This gave them enough leverage to create more complex works. Student were required to cite at least two sources for their work, preferably using Google Scholar[11], but security forums were also tolerated. Some opted for projects that combined security with parallelism. For example, two projects were focused on scanning network traffic. One team choose to perform full packet inspection, while the other team decided to use metadata and machine learning. It is easy to see how PDC and security would seamlessly work together here. For example a neural network used to solve the *Where's Waldo?* challenge would morph into something designed to detect suspicious events in a system's log. Similarly, a parallel cluster analysis algorithm originally designed to work with generic datasets would be used to identify patterns inside sets of forensics artefacts such as network traffic traces or file registry stores [22].

Students were offered support outside the classroom to complete projects that required the use of technologies that had not yet been covered. For example they were familiar with the concept of a GPU but had not used OpenCL before, or they needed guidance to select a way to process lots of data efficiently (ex: a simple clustering algorithm or a parallel implementation for Aho-Corasick). A list of some student projects that involved PDC can be seen below along with a short description:

- intercept network traffic using libpcap and scan for known rules or file types
- scan network traffic, extract metadata identifying protocols, users, applications, build host profile and detect anomalies
- compute entropy for executable files and detect encrypted or compressed content
- trace an application using LD_PRELOAD or Detours and scan any memory buffers allocated as executable memory
- use QEMU to extract the instruction trace of a running process and scan for known patterns
- monitor processes running in an operating system and group them into clusters based on predefined features
- create a client/server application that implements some logic found in botnets

Just like after PA, some students continued these projects as part of their bachelor thesis or presented them in conferences.

F. Final Examination

Students had to undergo a theoretical and practical examination at the end of the semester. In order to pass they had to have a score of at least 40% of the practical exam and a grand total of more than 50% from the rest of the activities (laboratories, projects, theoretical and practical examination). We choose this formula to stimulate students into engaging in weekly assignments but also keep them focused for the practical examination. The practical examination would last between 3-4 hours and would require students to solve problems pertaining to malicious code analysis either using existing tools or writing code. While we couldn't ask them to write complex HPC code within the interval, we included subjects that required knowledge of parallelism, threads and processes. For example, students were required to download and execute a benign sample that used persistence and anti-debugging techniques. The sample would copy itself across the filesystem and create a fixed number of processes from different file images. The processes would contain some trace of *maliciousness* (for example the string "I am a VIRUS" somewhere in the data section) and would regularly communicate with each other to make sure that the number of instances does not drop below a certain threshold (otherwise they would spawn more copies of themselves). Students would have to write code that disrupts the flow of the target processes, kills them all at once and removes any trace of the created files. Depending on the target operating system students could attach and read from the memory of the running processes or create crash dumps and investigate offline. Use of threads to parallelize the detection tasks was advised but not mandatory.

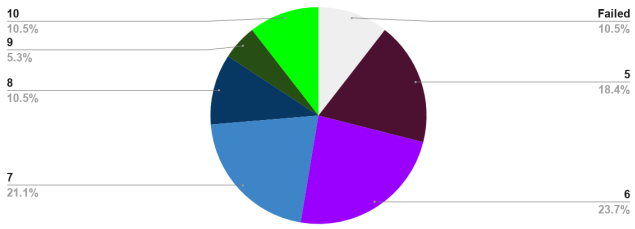

Fig. 2. Student scores for Malicious Code Analysis. Maximum is 10, minimum for passing is 5.

G. Evaluation of the Course

We measured student reception using a survey similar to the one used in the PDC course. The reaction of the students was overwhelmingly positive. 83% gave the lecture maximum points on the feedback form. Approximately 10% of the students failed to graduate the course and required an additional examination session in order to pass. On the other hand there were a small number of high-flyers that went above and beyond and achieved the maximum grade. There was a visible enthusiasm, similar to what we had encountered with the PA course. Students reacted positively to the mixed assignments and were eager to use their GPUs to solve problems like accelerating string matching and encryption algorithms. Programming using intrinsics instead of raw assembly made it easier to understand instruction level parallelism and provided a way for students to optimize existing code and compare the performance to that of other forms of parallelism (for example task parallelism using threads, OpenMP). Students also became adept at debugging their own applications and recognized assembly instruction patterns that corresponded to

[11]https://scholar.google.com/

performance-critical loops. This, in turn, made it easier later on when they had to recognize certain behaviours while analyzing malicious code. The interactive nature of Intel®Intrinsics Guide also made the transition from C/C++ to assembly easier. There were a lot more new topics and student assignments were interdisciplinary in nature. We believe this contributed to increased interest in the course and gave the students a broader perspective on the challenges they will encounter after graduation.

The main difficulties encountered by the students consisted in the large number of technologies used, initial unfamiliarity with assembly language and Windows programming (previous courses managed to provide an introduction and we gradually noticed a significant improvement, but students still reported most of their issues here).

Students were eager to work with vulnerable code and learn how to exploit it. Some of them competed in capture the flag (CTF) activities and hackathons organized by external parties.

V. Conclusions and Future Work

Over the last two years we perfected the Malicious Code Analysis course and we synchronized it with the rest of the curriculum. In the final stage of our endeavour we will make the course available to all the students enrolled at the Faculty of Information Systems and Cyber Security. At the end of the day the industry wants engineers that can understand today's complex computing infrastructures and can create optimal solutions within the scope of their assignments, and academia wants future graduate students who continue to push the state of the art forward in their respective fields and bring contributions to research projects.

In the current paper we've outlined the contents of our Parallel Architecture course and presented how our experience teaching PDC topics helped us improve other undergraduate courses and create a new one: Malicious Code Analysis. We believe that there is a lot of common ground between security and high performance computing and we will continue to develop our courses in close connection. We intend to introduce PDC-related case studies in other courses such as Cryptography and Security Testing of Information Systems.

As most software and hardware today displays some level of parallelism we consider that one cannot be a good security researcher or analyst if one does not have a good understanding of both fields. Interest in hybrid (CPU+GPU) solutions has increased recently. For example, we are not far from having functional anti-malware engines that use the GPU to perform scans. On the other side, there have been few examples in the past of malware that can run malicious code on the GPU [23] [24], but we expect to see more of that in the future [25]. As new architectures and threats emerge, we are determined to continue our efforts to educate future software engineers about the importance of PDC and cybersecurity and how they can be used together to great effect.

Both courses were developed in close collaboration with defense organizations and local industry. The professors and assistants that created and taught the lectures were associated with companies such as Intel, Amazon, Fitbit and Bitdefender.

References

[1] D. L. Burley, M. Bishop, S. Buck, J. J. Ekstrom, L. Futcher, D. Gibson, E. Hawthorne, S. Kaza, Y. Levy, H. Mattord *et al.*, "Cybersecurity curricula 2017," *Version 0.75 Report*, vol. 12, 2017.

[2] M. Carabaş, A. Drăghici, G. Lupescu, C.-G. Samoilă, and E.-I. Sluşanschi, "Integrating parallel computing in the curriculum of the university politehnica of bucharest," in *European Conference on Parallel Processing*. Springer, 2018, pp. 222–234.

[3] S. A. Bogaerts, "One step at a time: Parallelism in an introductory programming course," *Journal of Parallel and Distributed Computing*, vol. 105, pp. 4–17, 2017.

[4] R. R. Fogleman, "Information operations: The fifth dimension of warfare," *Defense issues*, vol. 10, no. 47, pp. 1–3, 1995.

[5] L. Brent. (2019) Nato's role in cyberspace. [Online]. Available: https://www.nato.int/docu/review/articles/2019/02/12/natos-role-in-cyberspace/index.html

[6] M. Murphy, "War in the fifth domain. are the mouse and keyboard the new weapons of conflict," *The Economist*, vol. 1, 2010.

[7] C. Leopold, *Parallel and Distributed Computing: A survey of Models, Paradigms and approaches*. John Wiley & Sons, Inc., 2001.

[8] D. Culler, J. P. Singh, and A. Gupta, *Parallel computer architecture: a hardware/software approach*. Gulf Professional Publishing, 1999.

[9] A. Williams, *C++ concurrency in action: practical multithreading*. Manning Publ., 2012.

[10] M. J. Flynn, "Some computer organizations and their effectiveness," *IEEE transactions on computers*, vol. 100, no. 9, pp. 948–960, 1972.

[11] T. J. Rolfe, "A specimen of parallel programming: parallel merge sort implementation," *ACM Inroads*, vol. 1, no. 4, pp. 72–79, 2010.

[12] K. I. Iourcha, K. S. Nayak, and Z. Hong, "System and method for fixed-rate block-based image compression with inferred pixel values," Sep. 21 1999, uS Patent 5,956,431.

[13] R. Deaconescu and V. Gosu, "Automatic assignment checker," https://github.com/rosedu/vmchecker, 2015.

[14] R. Velea, C. Ciobanu, L. Margarit, and I. Bica, "Network traffic anomaly detection using shallow packet inspection and parallel k-means data clustering," *Studies in Informatics and Control*, vol. 26, no. 4, pp. 387–396, 2017.

[15] P. Chitra and S. K. Ghafoor, "Activity based approach for teaching parallel computing: An indian experience," in *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2019, pp. 290–295.

[16] J. Gressick and J. B. Langston, "The guilded classroom: using gamification to engage and motivate undergraduates," *Journal of the Scholarship of Teaching and Learning*, vol. 17, no. 3, pp. 109–123, 2017.

[17] O. Or-Meir, N. Nissim, Y. Elovici, and L. Rokach, "Dynamic malware analysis in the modern era—a state of the art survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–48, 2019.

[18] S. Maitra and S. Madan, "Intelligent cyber security solutions through high performance computing and data sciences: An integrated approach," *IITM Journal of Management and IT*, vol. 8, no. 1, pp. 3–9, 2017.

[19] G. Vasiliadis and S. Ioannidis, "Gravity: a massively parallel antivirus engine," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2010, pp. 79–96.

[20] M. M. Masud, T. M. Al-Khateeb, K. W. Hamlen, J. Gao, L. Khan, J. Han, and B. Thuraisingham, "Cloud-based malware detection for evolving data streams," *ACM transactions on management information systems (TMIS)*, vol. 2, no. 3, pp. 1–27, 2008.

[21] R. Velea, Ş. Drăgan, and F. Gurzău, "Cpu/gpu hybrid detection for malware signatures for battery-powered devices using opencl," in *Recent Trends in Computer Applications*. Springer, 2018, pp. 139–152.

[22] B. Lagny, "Analysis of the amcache," *ANSSI - DFIRSummit*, 2019.

[23] R. McMillan, "Gaming company fined $1 m for turning customers into secret bitcoin army— wired," 2013.

[24] E. Ladakis, L. Koromilas, G. Vasiliadis, M. Polychronakis, and S. Ioannidis, "You can type, but you can't hide: A stealthy gpu-based keylogger," in *Proceedings of the 6th European Workshop on System Security (EuroSec)*, 2013.

[25] G. Vasiliadis, M. Polychronakis, and S. Ioannidis, "Gpu-assisted malware," *International Journal of Information Security*, vol. 14, no. 3, pp. 289–297, 2015.