(*faculty stamp*)

COURSE DESCRIPTION

1. Course title: GPU PROGRAMMING AND ARCHITECTURE	**2. Course code** GPU
3. Validity of course description: 2013/2014	
4. Level of studies: BSc programme	
5. Mode of studies: intramural studies	
6. Field of study: MAKROKIERUNEK	(FACULTY SYMBOL) RAU3
7. Profile of studies: general	
8. Programme:	
9. Semester: 6	
10. Faculty teaching the course: Institute of Electronics (RAu3)	
11. Course instructor: Tomasz Topa, PhD, Eng; Artur Noga, PhD, Eng	
12. Course classification:	
13. Course status: elective	
14. Language of instruction: English	
15. Pre-requisite qualifications: C/C++ programming basics, familiarity with CPU architecture and multi-threaded programming; an understanding of computer graphics algorithms would be useful but is not necessary, as the basics will be covered on the course.	
16. Course objectives: The aim of this course is to introduce the programming techniques required to develop general purpose software applications for graphics processor units (GPUs). The ATI/AMD and/or NVIDIA GPU hardware allows achieving computing power unavailable for traditional central processing units (CPUs). Using CUDA and OpenCL framework, the course will focus on the solution to common problems encountered while developing software applications on the GPU. This will include an introduction to the programming techniques required to take advantage of the architecture, as well as more advanced optimization methodologies needed to get maximum performance out of the computing platform.	

17. Description of learning outcomes:

Nr	Learning outcomes description	Method of assessment	Teaching methods	Learning outcomes reference code
1.	has an in-depth knowledge on GPU low-level programming concepts	test	lecture	
2.	has detailed knowledge on modern GPU hardware architectures	test	lecture	
3.	understands how the GPU hardware architecture differs from more traditional CPU architectures, and how this impacts on the approach to developing software for the platform	test	lecture	
4.	can describe design decisions in a GPU software development using appropriate diagrams and schematics	test	lecture	
5.	can identify the key programming methods and use the tools needed to develop software for the GPU platform	lab work	laboratory	
6.	can analyze the impact of the hardware architecture on the execution of the GPU application, and implement solutions that will optimize performance	lab work	laboratory	
7.	can analyze the effectiveness of the solution by means of testing and evaluation	lab work	laboratory	
8.	can cooperate with other people to design, develop, prototype and evaluate a GPU software, as part of a team	lab work	laboratory	

18. Teaching modes and hours

Lecture / BA /MA Seminar / Class / Project / Laboratory

Lecture: 15 h, Laboratory: 15 h

19. Syllabus description:

Lecture:

1. Overview of computer animation and graphics systems: video display devices, output primitives, three-dimensional geometric modeling and transformations, illumination and surface-rendering methods, the viewing pipeline

2. Introduction to GPU programming: GPU architecture, overview of parallelism model, arithmetic accuracy and rounding
3. GPU hardware: CUDA Core, Radeon Core, special function unit, load/store unit, texture unit, dispatch unit, streaming multiprocessor, raster operations processor, thread sequencer, thread/graphics procesing cluster, streaming processor array
4. GPU progrmming model: threads and thread hierarchy, thread assignemnt and scheduling, synchronisation and transparent scalability, stream computing, host and device interactions
5. Execution model: warps, scheduling and divergence
6. Device memory: global and shared memory, local memory, constant and texture memory, registers, memory hierarchy, memory latency
7. Performance optimizations: instruction performance, memory access patterns, global memory coalescence, local memory bank conflicts, optimization strategies, data prefetching, thread granularity
8. CUDA: tools and libraries: detailed description of CUDA API, compilation using nvcc, debugging, profiling, basic libraries, project assignment
9. OpenCL: introduction to OpenCL, differences comparing to CUDA, exploiting OpenCL for hardware not accessible from CUDA
10. Case studies: acceleration of image and video compression, MRI reconstruction, molecular visualization and analysis, computational electrodynamics, quantum chemistry, bioinformatics, signal processing, financial modeling, neural networks

Laboratory exercises:

1. GPU programming environment – installation, configuration, running and deploying applications
2. Data transfer and data caching
3. Memory access pattern
4. Launching kernels – thread cooperation, thread and device synchronization
5. Optimizing kernel code
6. Atomics
7. Concurrent transfer and execution
8. Mixing CUDA/OpenCL and rendering
9. Debugging and profiling kernel code

20. Examination: Lecture - test; Laboratory - positive grade required for each laboratory exercise

21. Primary sources:

1. J. Sanders, E. Kandrot, *CUDA by example. An introduction to General-Purpose GPU Programming*, Addison-Wesley Press, New York, 2010
2. D. Kirk, W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*, Morgan Kaufmann Press, 2010
3. B. Chapman, F. Desprez, G.R. Joubert, A. Lichnewsky, F. Peters, T. Priol, *Parallel Computing: From Multicores and GPU's to Petascale*, IOS Press, 2010
4. A. Karbowski, E. Niewiadomska-Szynkiewicz, *Programowanie równoległe i rozproszone*, Oficyna Wydawnicza Politechniki Warszawskiej, 2009
5. R. Farber, *CUDA Application Design and Development*, Elsevier, 2011
6. NVIDIA, Compute Unified Device Architecture Programming Guide, Version 4.0, 06/05/2011

22. Secondary sources:

1. Wen-mei W. Hwu, *GPU Computing Gems*, Elsevier, 2011
2. R. Fernando, M. J. Kilgard, *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*, NVIDIA corporation Press, 2003
3. R. S. Wright, N. Heamel, G. Sellers, B. Lipchak, *OpenGL SuperBible, Comprehensive Tutorial and Reference*, 5th edition, Addison-Wesley Press, New York, 2011
4. R. S. Wright, M. Sweet, *OpenGL. Księga eksperta*, Helion 1999
5. NVIDIA Corporation. OpenCL. Programming guide for the CUDA Architecture, v2.3, Feb. 2010

23. Total workload required to achieve learning outcomes

Lp.	Teaching mode :	Contact hours / Student workload hours
1	Lecture	15/5
2	Classes	0/0
3	Laboratory	15/10
4	Project	0/0
5	BA/ MA Seminar	0/0
6	Other	5/10
	Total number of hours	35/25

24. Total hours: 60

25. Number of ECTS credits: 2

26. Number of ECTS credits allocated for contact hours: 1

27. Number of ECTS credits allocated for in-practice hours (laboratory classes, projects): 1

26. Comments:

Approved:

…… *(date, Instructor's signature)*

…… (*date , the Director of the Faculty Unit signature)*