Introducing Parallel Computing Concepts in Computer System Related Courses

Han Wan[1,2], Xiaopeng Gao[1,2], Xiang Long[1], Bo Jiang[1]
School of Computer Science and Engineering[1], Honors College of Beihang University[2]
Beihang University
Beijing, China
{wanhan, gxp, long, jiangbo}@buaa.edu.cn

***Abstract*—All semiconductor market domains are converging to concurrent platforms. This trend has certainly led real challenge to develop applications software that effectively uses these concurrent processors to achieve efficiency and performance goals. This paper argues that the Computer System related courses are natural places to introduce the parallelism, and the earlier to parallel computing concepts will have a wide reach. This paper showed how digital logic classes can motivate topics from parallel computing through common logic structures. We provided an alternative view of the digital logic topics, including: binary representation of integers using decision tree with recursive thinking; use a carry look-ahead adder to show how sequential operations can be parallelized. Another part to introduce parallel concepts focused on write high performance code with specific emphasis on graphic processing unit (GPU). In our teaching experience, parallel pattern teaching has been confirmed to be a useful pedagogical method for teaching parallel concepts. Finally, we report course experience in teaching parallel computing injected course, which resulted in positive student feedback.**

Keywords—parallel computing; Architecture and Organization course; curriculum design; practice; assessment

I. INTRODUCTION

Processor manufacturers scaled performance by increasing the number of cores instead of the method of increasing clock frequency. The rapid advances in parallel system and the ever-greater need for computational speed have led to an increased demand for trained professionals with skills to solving challenging scientific problems. The rise of multi- / many-core processing has introduced new urgency to learning the parallel system architecture.

In the meanwhile, CS Curriculum 2013[1] has shifted parallel computing from elective status into the core. While several papers have explored how to introduce parallel topics in programming and algorithms courses, there has been little discussion of the changes needed in computer organization and architecture coursework to help students understand key issues about the behavior of parallel system. Bruce [2]emphasized concurrency as a natural way of modelling problems and integrated the concurrency into an introductory course. They limit the interactions between threads to avoid race conditions, students learn how to use concurrent processes to solve -problems in ways more natural. Joel [3] introduced parallelism into the CS2 (data structures) course. They added one week's worth of parallel material near the end of the course. They explained the multicore hardware, the concepts of threads and multithreading, and basic ideas of parallel design. Department of Mathematics and Computer Science in Longwood University [4] enumerated various software models and programming options to assist in integrating parallel software design techniques into the traditional software development training. Ref. [5] focused on parallel programming models on multicore systems. Their multicore course modules include software tool modules and application modules. The former introduced the overview of multicore architecture and principles of multicore programming, whereas the parallel design patterns were used to help students finding concurrency, algorithm structure, and implementation.

The mentioned courses organized parallelism topics focused on different knowledge areas, such as software design, data structures and algorithms, software environments and the hardware [6]. The demands for CS graduates are, for given problem, they should be able to: decompose the problem into sequential and parallel portions, recognize possible parallel approaches, and implement the efficient strategy. First, students need to know about the hardware organization, otherwise, understand the performance issues are difficult. Secondly, they need to choose an appropriate software environment, which can help implement the solution using the chosen approach. The implementation work including choose an appropriate data structure and find parallel algorithm to solve the problem.

This paper argues that the Architecture and Organization (AR) related courses are natural places to introduce the parallelism. To tackle more modern architectural issues, we proposed extending the idea about 'what the world is' to discuss with the student about 'why the world is like this'. The teaching philosophy included the following aspects: understand the relationship between the system's elements in the height of system investigated view; using historical perspective to study the evolution of key architecture technologies; and exploration of quantitative characterization of the program's performance based on the balance of the design principles.

We outline the changes made to the Computer Architecture related courses at Beihang University. The main portion of the course expansion focused on how to introduce parallel

978-1-5090-5920-1/17/$31.00 ©2017 IEEE

computing concepts in Digital Logic Course and how to introduce general purpose computing on graphic processing unit (GPU) using CUDA C [7]. Through examples and projects, student learned the basics of GPU architectures, along with optimization the execution of their programs using the GPU's global and shared memories, and they gained the familiarity with thread synchronization and experience in using atomic operations.

In our several years' experience, the parallel scientific computing required students harness many knowledge and skills accumulated in the courses such as Calculus, Physics, Biology, and Chemistry. Sometimes, students lack the required background and must learn new concepts "on the fly", which added additional stress to the already challenging course. The paper also argues that students should be taught to solve problems using parallel patterns, which are industry-standard best-practice strategies for parallel problem solving. To support such teaching, the paper present following assist modules: tiled parallel method; memory access pattern; data reuse; parallel computation patterns, such as convolution, reduction, and scan.

The paper presents evidence that this injection of parallelism into AR using our mentioned teaching philosophy has been successful. The student from different major joined this course reflected that the rewarding experience is vital importance for they prepare to enter the changing modern work.

Since 2013, we organized the workshop for student who want to develop application for GPU, as well as those who want to develop programming tools and future implementations. We also conducted a GLES project – a practical general purpose computing on GPU (GPGPU) optimizing compiler using data sharing and thread coarsening technique [8]. In this project, students who finished our course are involved to do automatic optimization GPGPU programming at compile time based on divergence analysis. This work was accepted by the 27th international workshop on languages and compilers for parallel computing.

Learners discussed with the lecturer about how to build a virtualized GPU computing platform in clustered systems in 2014. All GPUs in system are abstracted as virtualized ones to composed a resource pool. The original GPU application can be migrated to the virtualized GPU computing platform without any modification and can access any GPU within the resource pool without the need for programmers to deploy MPI programming explicitly for multi-node, multi-GPU applications. The application is free from the limitations of GPU resources on a single node and can access any available GPU resources in the cluster system indiscriminately. This helped in improving overall system's resource utilization and throughput.

II. BACKGROUND

A. Course Orientation

Computer Architecture and Organization course is a study of the evolution of computer architecture; this course also does important impact on the system design related hardware and software courses. It provides the knowledge of factors influencing the design of hardware and software elements of computer systems. Topics usually include the following content: fundamentals of computer design, measuring performance, quantitative evaluation, trends and new enabling technologies [9].

With the development of multi-core and many-core technology, this has brought the comprehensive change into the software and computing technology. The teaching changes are divided into two types: one is setting up new architecture course, whereas the other is adding the new framework into existing knowledge of computer architecture curriculum.

"Udacity CS344: Intro to Parallel Programming" [10] is a free online course created by Nvidia and Udacity. This course present the fundamentals of parallel computing using the CUDA programming model.

University of Illinois opened course ECE498 AL (Spring 2010) [11] considered programming massively parallel processors for general computation. And now ECE 408 - APPLIED PARALLEL PROGRAMMING emphasis on mapping computations to parallel hardware using efficient data structures. It provided paradigms for efficient parallel algorithms, and application case studies.

The modern application of large-scale computing problems exists, whose course included the following content: how to make use to the full potential of the parallel processor, considering the performance, efficiency, energy consumption and other factors.

Stanford University taught how to effectively program massively parallel processors using the CUDA C programming language since 2010 [12]. Students developed familiarity with the language itself and were exposed to the architecture of modern GPUs. Atomic operations in CUDA and the associated hardware were discussed. They also presented the performance considerations, including: memory coalescing, shared memory bank conflicts, control-flow divergence, occupancy, and kernel launch overheads.

CUDA Programming on NVIDIA GPU [13] is a course provided by University of Oxford. This is a hands-on course for students to learn how to develop applications to run on NVIDIA GPUs using the CUDA programming environment. The practices focused on launching a kernel, copying data to/from the GPU; thread block size optimization, multi-dimensional memory layout; thread synchronization; overlapping computation and communication.

University of Wisconsin [14] opened "ME964: High-Performance Computing for Applications in Engineering". CUDA execution configuration, GPU scheduling issues, and control flow were introduced. And then it elaborated tiling programming pattern and CUDA optimization issues. This course also provided several homeworks to help students grasp the basic software design patterns for high performance parallel computing, such as matrix multiplication, 2D convolution, and parallel scan.

There are many other universities had introduced the parallel knowledge based on the CUDA environment [15][16]. Our course at Beihang University opened orienting students from all the majors. The necessary structural knowledge is

presented to help students in understanding the basic structure and latest achievements of the system. We injected the parallel computing into AR course since 2009, and we chosen CUDA as the study platform since its similarity to the C language. We had adjusted the content based on students' feedback each round. In conjunction with theory of teaching curriculum, lab practices are requested to assist the understanding the parallel content.

B. *Automatic Optimization GPGPU Programming at Compile Time*

Since CUDA exposes too many architectural related details, developing and maintaining a highly efficient CUDA program is difficult. We introduced CUDA-lite [17] work on automatic optimizing GPGPU programming after the performance tuning content. CUDA-lite takes programmer's annotation to perform GPU memory hierarchy optimization. Inspired by its work, student discussed how to apply data sharing and thread coarsening optimization.

In this part, students need to further deep understand the divergence analysis in the GPGPU programming. Since CUDA implements a single program multiple data (SPMD) model, only the statements that are data dependent on divergent variables may produce different results during kernel execution.

III. Parallel Computing Concepts Elabrated in Digital Logic Course

We offered an alternative view of the digital logic topic, but did not discuss the parallel and distributed computing ideas in much detail.

A. *Number Representation using Decision Tree*

The binary representation of integers is one of the first topics need to be introduced in the digital logic.

In class, consider pick a number x from the set {0, 1, 2, …,7} ([0,7]), then derive the 3-bit binary representation of x.

- First, determine whether x is in the top half [0,3] or the bottom half [4,7]. If x in [0,3], record a '0' to indicate the bottom half; else, record a '1' to indicate the top half.
- Record the next level of x as 0 or 1 depending on whether x is in the bottom or top half of its level.
- Finally, we have presented the binary representation of x as a sequence of decisions.

As shown in Figure 1, the binary decision tree represents a 3-bit numbers in the range [0,7], and the red path indicates the binary representation of number 5. It also showed the node levels and the bit positions corresponding to edge labels.

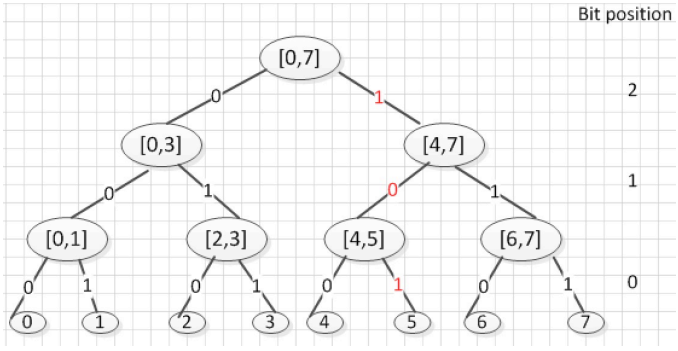

Fig. 1. Binary decision tree represents all possible outcomes for set [0,7]

This tree could be used as a basis for several concepts in digital logic, parallel and distributed computing (PDC). When dividing the initial range into the bottom and top halves, which formed the left and the right subtrees. And in the same way (recursively), the subtrees themselves were constructed.

This manner of expressing a large problem in terms of smaller instances of the same problem is an easy way to introduce divide-and-conquer paradigm.

B. *Carry-look-ahead Adder*

When discuss the combinational circuits in the digital course, we introduced the standard ripple carry adder as shown in Figure 2. Students will see the bottleneck of the ripple-carry adder's speed is the sequential generation of carry bits. And the important design principle of improving the performance of a module is reducing this critical path.

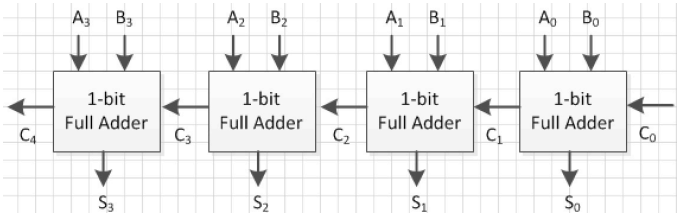

Fig. 2. A 4-bit ripple carry adder

Then we use a carry look-ahead adder to show how sequential operations can be parallelized as shown in Figure 3.

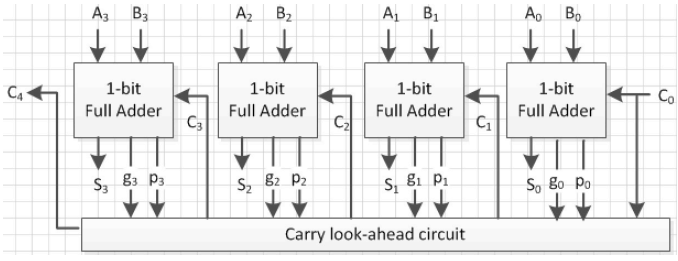

Fig. 3. A 4-bit carry look-ahead adder

We also introduced the idea of parallel prefix, which is used in numerous applications.

IV. Injecting Parallel Computing into AR Course

Since the focus of our AR course is the advanced architecture course, we incorporated parallelism throughout the following facets [21]: classroom sessions, assignments, and lab exercise.

A. Classroom Sessions

In Spring 2014, we referenced Illinois ECE 408 [18][19], and combined with our own experience and teaching philosophy, used these sessions as follows:

- The architecture about CPU and GPU's evolution: we emphasized using historical perspective to study the evolution of key architecture technologies. The performance of single-core and Moore's law were introduced. Students need to know the superscalar architecture how to implicit instruction level parallelism, and the pipeline technique. Taught the knowledge of the power wall and memory wall. Then we introduced the multi-core and many-core architecture. From introduction of the fixed function graphics pipeline to programmable graphics pipeline and then 3D graphic pipeline. We finally discussed the different design between the CPU and GPU, and compared their performance.

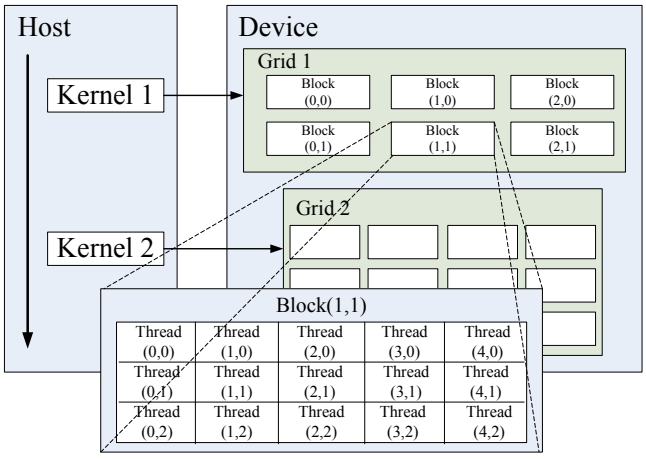

Fig. 4. CUDA Programming Model [7]

- General purpose computing on GPU against the traditional graphic programming: Computed Unified Device Architecture, evolution of CUDA-Enabled GPUs with different compute abilities. Parallel thread organization (thread, thread block, grid and kernel) was elaborated as shown in Figure 4.
- How the GPU work together with CPU, the basic API function, such as device memory allocation and host-device data transfer. The simple first CUDA program to elaborate how to compose the host code, device code and kernel launch. Here we emphasized the importance of problem decomposition, student should know the work for each thread and then how to make use of the parallelism.
- Kernel based parallel programming knowledge, including streaming processor, streaming multiprocessor, thread scheduling, and control divergence.
- Matrix multiplication from the CPU version to GPU version; GPU memory model study; Improve the matrix multiplication using tiled parallel algorithm, using the shared memory to reuse the global memory data, and how to handle the boundary conditions.
- Performance considerations: DRAM burst and memory coalescing; learn to find out the un-coalesced accesses; elaborated the resources restrictions (registers, shared memory, thread block slots, and thread slots), and how the resource affect the parallel execution; gave the examples such as reduce the control divergence, data prefetching to increase the independent instruction amount between the memory access instruction and the instruction which use the access result, unroll loop to reduce the processor's instruction bandwidth consume.
- Floating computing in CUDA: how to represent the float using 5-bit IEEE format; parallel algorithm consideration about float computing.

B. Assignments

Students' assignments are used to consolidate the knowledge taught in the class session. These assignments usually composed by several choice questions. And the course center web evaluates the results immediately. The correct rate helped us to find out which knowledge point should be explained more detailed on next lecture.

- Basic questions about the GPU architecture, CUDA programming model, the device and host transfer CUDA API.
- Small examples with the question about kernel execution, thread warps, and control divergence.
- Compute the thread number for each SM under the hardware resources restrictions.
- Tiling technique's impact on the system, such as memory bandwidth usage, DRAM burst, and the control divergence will happen when handling the boundary conditions.
- Memory access evaluation, how shared memory accesses varied according to the tile size.

C. Labs

We provide three types GPU for students to do the labs, including GTX-280, GTX-480 and C1060. Students finished the lab to learn the hardware resources amount, principle of CUDA parallel programming and the performance tuning:

- Building the local CUDA environment, learn to use the CUDA SDK.
- Analysis the MatrixMul program, learn about the skeleton of the CUDA programming, know the basic

device and host communication, and study its programming model.

- Compare the hardware resources amount between offered GPUs. Tuning the program on the use of the registers, shared memory, thread block slots, and thread slots, in order to know the allocation strategy's impact on the system performance.
- Learn how to use the timing API, this help in the later performance evaluation.
- Students combine the major and the course learning: choose one application, describe the algorithm, and break it down into parallel-able part and sequential part. Then speed up the parallel-able part using CUDA, and tuning the performance using the techniques, such as tiling, making use of shared memory, reduce the control divergence and coalescing global memory access.

V. LEAN FROM TEACHING

The computer architecture course for undergraduate often focus on 'what the world is', our AR course for graduate paid more attention to elaborate 'why the world is like this'. From our teaching practice, we found the following teaching philosophy do help students' comprehension.

A. Historical Perspective to Study the Evolution

As we mentioned in last section, we first elaborated the architecture about CPU and GPU's evolution. When we introduced the GPU architecture, students need to know the control logic and memory model differences between CPU and GPU. This helped them in understanding the reason about the huge gap in the throughput of peak floating computation between CPU and GPU.

The introduction about the GPU's evolution from fixed function graphics pipeline was also necessary. When they can understand the graphics processor inheritance, the advantages and disadvantages of the current major computing model can be understood, especially help them to comprehend the modern GPU design philosophy – "massive parallel threads, a relatively small cache, and increased memory bandwidth". This also helped to grasp the future GPU development trend.

B. Balance Design based on Quantitative Characterization

We used the quantitative characterization to introduce the influence of different resources allocation on the performance.

For example, the increase usage of the register for each thread may result in the decrease number of thread block parallel execution, which always called "performance cliff".

When tuning the performance of Matrix multiplication, each thread can compute two elements in the result matrix instead of just compute one element. This programming improvement may decrease the access number to the global memory and increase the independent instructions in the prefetching ploy. But on the other hand, this improvement may use more registers and shared memory, which may all lead to performance cliff.

To sum up, students need to do balance design from the system-investigated view using quantitative characterization.

C. Parallel Patterns

Since the parallel scientific computing required students harness many knowledge and skills accumulated. We tried illustrating the parallel computing pattern during Spring 2014. These patterns are best-practice strategy that has found to be frequently useful in solving problems. And we found it was a good investment of our limited classroom time.

1) Convolution Computation: Learn the basic 1D and 2D convolution operation; using tiled technique to parallel convolution; using constant memory to store the mask; pay attention to the boundary handling.

2) Reduction: Learn reduction operation such as Max, Min, Sum and Product.

a) Basic reduction: if the thread id is even number, it will in charge of does the operation, its inputs from current address and the address plus step. For the first time, the step is 1. And for each iteration, the step will be equal to 2 times of current value. In the meanwhile, the working thread number is half of last iteration.

b) A better reduction: the operation compacts to the first half of the threads. For each working thread, its inputs from current address and the address plus step. For the first time, the step equals to half the array size. And for each iteration, the step will be equal to half of the current value.

c) Compare the two implementations, especially focus on the control divergence.

3) Scan (prefix-sum): Sequential algorithm will be compare with the parallel algorithm. Two phased balanced tree is elaborated, which can share the intermediate results and decrease the control divergence.

VI. ASSESSMENT AND CONCLUSION

Peter J.Denning has pointed out the great principles of computing in [20], which can be grouped into following categories: computation, communication, coordination, recollection, automation, evaluation and design. For our parallel computing topic, after the lectures the student should be able to:

- Understand the basic architecture of GPU – have the practical knowledge of how GPU programs operate and how they can be utillized for high performance applications (computation, recollection).
- The communication in the GPU - how thread exchange information and coordination between CPU and GPU, and how to coordinate work among multiple GPUs (communication).
- Identify and solve a computational problem with parallel algorithm design and apply parallel programming interface features (design, automation).
- Predict the performance of the system based on the hardware resources' usage and apply common parallel

techniques to improve performance given hardware constraints (evaluation).

We design series of lectures around each principle and find examples to exemplify it.

We report our experience in the parallel computing during the spring 2014. This course is positioned as an elective class for students.

A. Assignments

One way that we assess student learning during this parallel computing course is through the assignment released after the classroom session. The grade distribution of each assignment is list in Table 1.

TABLE I. GRADE DISTRIBUTION IN SPRING 2014

Grade Range	% of Students		
	Assignment 1	*Assignment 2*	*Assignment 3*
A 84% to 100%	50%	20%	60%
B 75% to 83%	40%	20%	0
C 63% to 74%	10%	10%	0
D 50% to 62%	0	30%	10%
F Below 50%	0	20%	30%

Detailed analysis had been done to profile student learning. Specially to find out which knowledge points exist inaccurate understanding, and should be reviewed in the classroom session.

B. Pattern Practice

In Spring 2014, we argued that students should be taught to solve problems using parallel patterns, which are best-practice strategies for parallel problem solving. To support such teaching, "convolution, reduction and scan" are presented in the following way: first introduce the sequential algorithm, then find out its parallelism, implement it using CUDA, and finally tuning the performance.

Given this step-by-step progress in experience, and maturity of the two groups (Spring 2014 and Spring 2013), we believe the pattern teaching had a more significant effect in helping our students understand parallelism according to the grade and feedback.

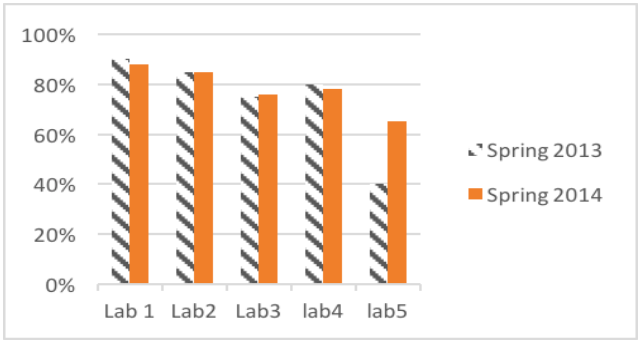

Fig. 5. Degree of lab completion in Spring 2013 compared with spring 2014

Course in Spring 2013 and Spring 2014 had similar labs (contained same knowledge points), as mentioned in section 3.3. As shown in Figure 5, the first 4 labs' completion degrees were very close, but the last integrated lab, there was a huge difference in the completion degree between two groups. From the lab reports, we discovered that Spring 2014 group, who learned the parallel patterns, had more clear understanding in the task decompose and quicker implementation than Spring 2013 group.

C. Students' Feedback

The first part questions related to the course contents, nearly 77% students could analyze and implement common parallel algorithm patterns in CUDA.

The second part questions focus on the lab sections. Most students gave positive feedback about the practical work done in the lab sessions. They like the lab difficulty from the easier to the more advanced. And they announced that the hands-on programming based on the skeleton programs is welcomed. The lab sessions use a parallel debugger to identify code defects and use a parallel profiler to identify performance bottlenecks in the code, can help most students to learn and understand the major types of hardware limitations that presented in class.

The third part questions are about the architecture research session. Most students denoted that this session helped them lots in knowing the common architecture research methodology with tool, such as simulation methodology and Pin tool.

The overall valuation of the course turned out positive, but the flood of information still overwhelmed less than 20% students.

Based on our students' assignments, labs and comments, we thought that our injection of parallel computing into AR was successful.

D. Conclusions

With the advent of parallel computing, CS departments must face the question of how to integrate the parallel knowledge units into their curricula. In this paper, we have argued that AR related courses are natural places to introduce parallelism.

Since the digital logic course lays the foundation of AR course, the earlier to parallel computing concepts will have a wide reach. This paper showed how digital logic classes can motivate topics from parallel computing through common logic structures.

In our GPU based parallel training practice, classroom sessions, assignments and hands-on labs are used. For the classroom session, we use the parallel pattern assist in the teaching, which allow student to quick grasp the essential parallel concept. On the other hand, these patterns are likely to remain useful for their professional futures.

Lastly, we conducted several projects after the course since 2012. Those projects showed that students learned the necessary domain knowledge to solve the identified problem.

During the discussions, they could motivate the problems and approach to solve them. Furthermore, they could identify the limitaions of the solutions and future directions. In the presentations, they could explain the experiments. The most important is that they could divide the responsibilities among teammates and support each other towards success.

ACKNOWLEDGMENT

This work is supported by China Scholarship Council (No.201406025114) and the Teaching Research Funding in Honors College of Beihang University (2017).

REFERENCES

[1] The Joint Task Force on Computing Curricula Association for Computing Machinery (ACM), IEEE Computer Society, "Computer science curricula 2013," DOI: 10.1145/2534860, December 20, 2013.

[2] Kim B. Bruce, Andrea Danyluk, and Thomas Murtagh, "Introducing Concurrency in CS 1," SIGCSE'10, Milwaukee, Wisconsin, USA, pp.224-228, March 10–13, 2010.

[3] Joel C. Adams, "Injecting Parallel Computing into CS2," SIGCSE'14, Atlanta, Georgia, USA, pp.277-282, March 5–8, 2014.

[4] John R. Graham, "Integrating Parallel Programming Techniques into Traditional Computer Science Curricula," SIGCSE Bulletin, vol. 39, num. 4, 2007 December.

[5] Pangfeng Liu, Greg C. Lee, Jenq-Kuen Lee, and Cheng-Yen Lin, "Innovative System and Application Curriculum on Multicore Systems, " WESE 2011 (ESWeek 2011), pp.25-31.

[6] Richard Brown, Elizabeth Shoop, Joel Adams, Curtis Clifton, "Strategies for Preparing Computer Science Students for the Multicore World," ITiCSE-WGR'10, pp.97-115, Bilkent, Ankara, Turkey, June 26–30, 2010.

[7] Nvidia CUDA Zone [EB/OL], https://developer.nvidia.com/cuda-zone.

[8] Zhen Lin, Bo Jiang, Han Wan and Xiaopeng Gao, "GLES: A Practical GPGPU Optimizing Compiler Using Data Sharing and Thread Coarsening," the 27th International Workshop on Languages and Compilers for Parallel Computing. September 15-17, 2014. Intel Corporation, Hillsboro, OR.

[9] John L. Hennessy, and David A. Patterson, Computer Architecture: A Quantitative Approach. ISBN-13: 978-0123838728. Morgan Kaufmann; 5th edition (September 30, 2011).

[10] Udacity CS344: Intro to Parallel Programming [EB/OL], https://cn.udacity.com/course/intro-to-parallel-programming--cs344/.

[11] ECE 498 AL : Applied Parallel Programming [EB/OL], https://courses.engr.illinois.edu/ece498al/.

[12] CS 193G: Programming Massively Parallel Processors with CUDA [EB/OL], https://itunes.apple.com/us/itunes-u/programming-massively-parallel/id384233322.

[13] Course on CUDA Programming on NVIDIA GPUs [EB/OL], http://people.maths.ox.ac.uk/~gilesm/cuda/.

[14] ME964: High Performance Computing for Engineering [EB/OL], http://sbel.wisc.edu/Courses/ME964/.

[15] Engineering Tool IV - Introduction to GPU Programming [EB/OL], http://www.cse-lab.ethz.ch/index.php/teaching/42-teaching/classes/576-etvgpufall2013.

[16] Existing University Courses [EB/OL], https://developer.nvidia.com/educators/existing-courses.

[17] S. Ueng and M. Lathara, "CUDA-lite: Reducing GPU programming complexity," International Workshop on Languages and Compilers for Parallel Computing, 2008.

[18] Applied Parallel Programming [EB/OL], http://courses.engr.illinois.edu/ece408/.

[19] D. B. Kirk and W.-m.W. Hwu, Programming massively parallel processors: a hands-on approach. Morgan Kaufmann Publishers, Burlington, MA, 2010. ISBN 0123814723.

[20] Peter J.Denning, "Great Principles of Computing, " Communication of ACM, Nov. 2003, 46(11), 15-20.

[21] Han Wan, Xiaopeng Gao, Xiang Long and Yi Li, "Injecting parallel computing into Architecture and Organization course", EduHPC 2015-Workshop on Education for High-Performance Computing, http://grid.cs.gsu.edu/~tcpp/curriculum/?q=EduHPC15_Technical_Program