Subject Code: 01CO0305

Subject Name: GPU Computing

M. Tech. Year - II

Objective: Almost, all modern processors are multi-core, and in order to make effective use of them, you need to run multiple threads. When accelerators, such as GPUs, are available, then the number of threads needs to be even larger. This course teaches you how to organize the computations of the threads so that they work together and performs the required computations efficiently, making good use of the available hardware resources. We will focus on using GPUs for general purpose computing, rather than for graphics.

Credits Earned: 4 Credits

Course Outcomes:

At the end of the course, students will be able to:

- Define terminology commonly used in parallel computing, such as efficiency and speedup.
- Describe common GPU architectures and programming models.
- Implement efficient algorithms for common application kernels, such as matrix multiplication.
- Given a problem, develop an efficient parallel algorithm to solve it.
- Given a problem, implement an efficient and correct code to solve it, analyze its performance, and give convincing written and oral presentations explaining the achievements.

Teaching and Examination Scheme

Teaching Scheme (Hours)			Credits	Theory Marks			Tutorial/ Practical Marks		Total Marks
Theory	Tutorial	Practical		ESE (E)	IA	CSE	Viva (V)	Term work (TW)	
3	0	2	4	50	30	20	25	25	150

Content:

Sr. No.	Topics	Contact Hours
1	**Introduction**: History, GPU Architecture, Clock speeds, CPU / GPU comparisons, Heterogeneity, Accelerators, Parallel Programming, CUDA OpenCL / OpenACC, Kernels Launch parameters, Thread hierarchy, Warps/Wavefronts, Threadblocks/Workgroups, Streaming multiprocessors, 1D/2D/3D thread mapping, Device properties, Simple Programs	13
2	**Memory:** Memory hierarchy, DRAM / global, local / shared, private / local, textures, Constant Memory, Pointers, Parameter Passing, Arrays and dynamic Memory, Multi-dimensional Arrays, Memory Allocation, Memory copying across devices, Programs with matrices, Performance evaluation with different memories	7
3	**Synchronization:** Memory Consistency, Barriers (local versus global), Atomics, Memory fence. Prefix sum, Reduction. Programs for concurrent Data Structures such as Worklists, Linked-lists. Synchronization across CPU and GPU **Functions:** Device functions, Host functions, Kernels functions, Using libraries (such as Thrust), and developing libraries.	10
4	**Support:** Debugging GPU Programs. Profiling, Profile tools, Performance aspects **Streams:** Asynchronous processing, tasks, Task-dependence, Overlapped data transfers, Default Stream, Synchronization with streams. Events, Event-based- Synchronization - Overlapping data transfer and kernel execution, pitfalls.	8
5	**Case Studies:** Image Processing, Graph algorithms, Simulations, Deep Learning	5
6	**Advanced topics:** Dynamic parallelism, Unified Virtual Memory, Multi-GPU processing, Peer access, Heterogeneous processing	5
	Total Hours	48

Reference Books:

1. David Kirk and Wen-mei Hwu, Programming Massively Parallel Processors: A Hands-On Approach, 2nd Edition, Publisher: Morgan Kaufman, 2012, ISBN: 9780124159921.
2. 2. Shane Cook, CUDA Programming: A Developer's Guide to Parallel Computing with GPUs, Morgan Kaufman; 2012 (ISBN: 978-0124159334)Wilkinson, M.Allen, Parallel Programming Techniques and Applications using networked workstations and parallel computers, Prentice Hall, 1999

List of Practical's:

1. Write CUDA code to compute the squares of the first N integers.
2. Write CUDA code to determine the following:
   i. data transfer bandwidth from host to device
   ii. data transfer bandwidth from device to host
   iii. data transfer bandwidth from host to device using pinned memory
3. Implement matrix multiplication on the CPU and GPU(without using shared memory), and compare their relative performances in terms of GFlop/s and report your performance results.
4. Write code for matrix multiplication using shared memory and compare its performance with CPU code. Determine the best tile-size for matrix multiplication using shared memory for your GPU machine.
5. Implement the 1-D convolution kernel and compare the performance with and without shared memory.
6. Perform 2-D convolution on an n x n matrix with an m x m mask and determine the number of halo cells required for it.
7. Implement 2-D convolution with data in shared memory. Also, analyze the reduction in bandwidth from use of shared memory in 2-D convolution.
8. Write code for histogramming without atomic operations and experiment with it to find out the fraction of times that it gives incorrect results and report it.
9. Rewrite program 8 so that it uses atomic operations, and compare its performance with it and report it.
10. Write code to perform reduction using atomic operations on global memory. (ii) Rewrite this code to use shared memory in order to reduce the overhead of atomic operations. (iii) Compare the performances of the above two with the reduction algorithm we had discussed earlier.
11. Write code with different threads performing atomic operations on different, but adjacent, locations in memory. Compare the performance with different data sizes.