GUJARAT TECHNOLOGICAL UNIVERSITY

Master of Engineering
Subject Code: 3730210
Semester – III
Subject Name: Compiler for HPC

Type of course: Elective

Prerequisite: Computer Organization and Architecture, Basics of Compiler Design

Rationale: Optimizing compilers play a critical role in modern computer systems ranging from mobile devices to supercomputers. Compilers can optimize for performance, power consumption and/or code size. Practically all computer scientists and engineers may benefit for a deep knowledge of compiler optimizations: programmers and application optimizers write programs that are better optimized by the compiler, computer designers design hardware features that are easy to use by compilers, and finally compiler writers develop new compiler optimizations. This course covers optimizations and aspects of the compiler back-end and middle-end such as: data-flow analysis, control Flow analysis, instruction level parallelism, memory hierarchy optimizations, data level parallelism and thread level parallelism.

Teaching and Examination Scheme:

Teaching Scheme			Credits	Examination Marks				Total Marks
				Theory Marks		Practical Marks		
L	T	P	C	ESE (E)	PA (M)	ESE (V)	PA (I)	
3	0	0	3	70	30	0	0	100

Content:

Sr. No.	Content	Total Hrs	% Weightage
1	**High Performance Systems,** Structure of a Compiler, Programming Language Features, Languages for High Performance.	6	12
2	**Data Dependence:** Data Dependence in Loops, Data Dependence in Conditionals, Data Dependence in Parallel Loops, Program Dependence Graph. **Scalar Analysis with Factored Use-Def Chains:** Constructing Factored Use-Def Chains, FUD Chains for Arrays, Induction Variables Using FUD Chains, Constant Propagation with FUD Chains, Data Dependence for Scalars. Data Dependence Analysis for Arrays, Array Region Analysis, Pointer Analysis, I/O Dependence, Procedure Calls, Inter-procedural Analysis.	10	21
3	**Loop Restructuring:** Simple Transformations, Loop Fusion, Loop Fission, Loop Reversal, Loop Interchanging, Loop Skewing, Linear Loop Transformations, Strip-Mining, Loop Tiling, Other Loop Transformations, and Inter-procedural Transformations. **Optimizing for Locality:** Single Reference to Each Array, Multiple References, General Tiling, Fission and Fusion for Locality.	10	21
4	**Concurrency Analysis:** Concurrency from Sequential Loops, Concurrency from Parallel Loops, Nested Loops, Round off Error, Exceptions and Debuggers. **Vector Analysis:** Vector Code, Vector Code from Sequential Loops, Vector Code from For all Loops, Nested Loops, Round off Error, Exceptions, and Debuggers, Multi-vector Computers.	10	21
5	**Message-Passing Machines:** SIMD Machines, MIMD Machines, Data Layout, Parallel Code for Array Assignment, Remote Data Access, Automatic Data Layout, Multiple Array Assignments. **Scalable Shared-Memory Machines:** Global Cache Coherence, Local Cache	10	21

	Coherence, Latency Tolerant Machines.		
6	Recent trends in compiler design for high performance computing and message passing machines and scalable shared memory machine.	2	4
	Total	48	**100%**

After learning the course the students should be able to:

Sr. No.	**CO statement**	**Marks % weightage**
CO-1	Be familiar with the structure of compiler	20%
CO-2	Understand the performance characteristics of modern processors	40%
CO-3	Have experience with algorithms for automatically taking advantage of SIMD, SIMT, and MIMD parallelism	40%

Distribution of marks weightage for cognitive level

Bloom's Taxonomy for Cognitive Domain	**Marks % weightage**
Recall	15
Comprehension	20
Application	15
Analysis	20
Evaluate	20
Create	10

Reference Books:

1. Muchnick, Steven S, *Advanced compiler design implementation*, Morgan Kaufmann, cop. 1997. ISBN: 978-1558603202
2. Michael Wolfe, *High-Performance Compilers for Parallel Computing*, Pearson
3. Aho, Alfred V, *Compilers : Principles, Techniques, and Tools*, Addison-Wesley, cop. 2007. ISBN: 9780321486813
4. Allen, Randy; Kennedy, Ken, *Optimizing Compilers for Modern Architectures : A Dependence-Based Approach*, Morgan Kaufmann Publishers, cop. 2002. ISBN: 1-55860-286-0

Practical List:

1) Setup LLVM on your machine. You should now have three directories (SimplePass, CellularAutomata, MysoreScript), one for each example.In each of these, you will find two build directories.
2) The SimplePass example must be modified to count instructions per basic block.
3) MysoreScript is a very simple language that provides a JavaScript-like model. You should improve the system using **improved dispatch tables**, replacing the linked list. Try adding either a sparse tree or inverted dispatch tables (where each selector has a class-to-method mapping, rather than each class having a selector-to-method mapping) and modify the compiler to do lookups inline, rather than calling out to C code. Whichever option you pick, show some example code where it gives a performance increase and be prepared to justify whether this is representative.
4) This is a simple compiler for a domain-specific language for generating cellular automata. The language itself is intrinsically parallel—you define a rule for updating each cell based on its existing value and

neighbours—but the compiler executes each iteration entirely sequentially, one cell at a time. Introduce the following parallelism into this system.

Vectorised implementation: The current version is not amenable to automatic vectorisation because the edge and corner implementations are not the same as the values in the middle. Modify the compiler to generate three versions of the program: one for edges, one for corners, and one for the middle. Make the edge and middle implementations simultaneously operate on 4 (or more) cells by using vector types in the IR. Be careful with the global registers!

List of Open Source Software/learning website:

- http://llvm.org/docs/LangRef.html