Instructor:  
Dr. Pedro Diniz, email pedro@isi.edu

Lectures:  
Wednesday, 3.30 – 6.20 PM, Grace Ford Salvatori (GFS) 229

Office hours:  
Wednesday, 2.30 to 3.30 PM (just before class), Office: SAL 234

Teaching Assistants: Due to low enrollment a TA is not likely to be assigned to this class.

Description:  
This course covers compiler analysis and program transformation topics and their interplay with advanced high-performance computing architectures such as parallel and multi-core architectures. The course covers advanced topics not addressed in CSCI 565 (Compilers Design) such as array data-dependence analysis, instruction scheduling/prefetching and vectorization and parallelization. The goal of this class is to give the students breadth knowledge on compiler analysis and transformation that exploit advanced architectural features for high-performance computing.

Prerequisites:  
Students should be familiar with the concepts from compiler design and implementation covered in the CSCI565 (Compiler Design) class, namely, data-flow analysis, control-flow analysis and program execution concepts such as address space and instruction execution. To carry out their programming assignment students must master programming languages such as C or C++ and have access to USC’s computing infrastructure. In specific cases the student may use other computing facilities upon approval to ensure they meet class goals.

Assignments:  
We expect to have 3 individual homework assignments throughout the course in addition written in-class exam. Homeworks are due at the beginning of the class on the due date. Late homeworks will not be accepted as the solutions are posted within hours of the end of the class. Hardcopies of homeworks only (exception for remote students). In extreme circumstances please ask a fellow student to bring your homework in a sealed envelope. Homeworks are organized and structured as preparation for the midterm and final exam, and are meant to be a good studying material as the level of difficulty of the midterm and final will be identical to the homeworks.

There will also be 2 individual programming projects in this class. The goal of these projects is to have the students experiment with very practical aspects of high-performance computing program analysis and transformations. These projects will cover topics we study in class and will the vast code basis available at the class web site you should have no problem completing each of them in under a week.

Grading:  
We will assign 15% to 3 homeworks (some might have larger percentages due to difficulty), 45% for the two programming projects (20% first, 25% second), and 40% for a written test offered in the last week of class.
**Important Note on Grading:**

*It is absolutely against university policy for faculty to grant appeals for grade change for any reasons other than mistakes in grading. The student's request is completely inappropriate and - even if faculty is sympathetic - it is not possible within the rules to grant the requested change.*

**Textbook:**

There is really no textbook that covers all the topics in this class. The volumes shown below, some of which are clearly dated and not easily available at any bookstore, covers most of the material in this class. You do not have to acquire any of these volumes unless you would like to have it for future reference. Of these I would recommend only the first volume as it covers a wider range of topics. As such, I will, occasionally, distribute supplemental material in class to cover topics that are not easy to find elsewhere.

*Title: Optimizing Compilers for Modern Architectures: A Dependence-based Approach*
*Author: R. Allen and K. Kennedy*
*Publisher: Morgan-Kaufmann Publishers*
*ISBN: 978-1558602861*
*Year: 2001*

*Title: High-Performance Compilers for Parallel Computing*
*Author: M. Wolfe*
*Publisher: Addison-Wesley*
*ISBN: 978-0805327304*
*Year: 1995*

*Title: Advanced Compiler Design and Implementation*
*Author: S. Muchnick,*
*Publisher: Morgan-Kaufmann Publishers*
*ISBN: 1-558600-320-4*
*Year: 1997*

**Class Material:**

Please gain access to course-related material on-line at the class web site (http://www.isi.edu/~pedro/Teaching/CSCI595-Fall10/). You can download course notes, check your grades, post questions, participate in discussion groups, keep current with class announcements, and do other useful things.
Below is a tentative description of the contents of each lecture. We may change the order to accommodate the materials you need for the projects.

**Lecture 1: Introduction and the Role of Compilers in High-Performance Computing**
- Motivation: Complexity of Computing Architectures
- Basic Architectural Elements: Caches, Multi-core, Pipelines, multi-processors and its organization.
- Basic Programming Models: Vectorization, Parallelism – shared and distributed memory
- Example of the impact of compiler transformations
- Overview of the use of advanced compiler techniques and code transformations.

**Lecture 2: Data Dependence Analysis: Basic Blocks and Loops**
- Data dependence for scalars; types of dependences
- Loops and iteration space definitions
- Array region and dependence representations
- Loop-carried dependences and dependence vectors and distances
- Simple dependence tests for loops

**Lecture 3: Loop Transformations for the Memory Hierarchy**
- Introduction and Rationale: Registers versus Caches
- Loops and Locality: Temporal and Spatial Locality
- Scalar Replacement and Register Pressure
- Loop unrolling, Unroll-and-Jam and Tiling in Scientific Codes
- Case Study: Matrix-vector and Matrix-vector Multiplication

**Lecture 4: Instruction Scheduling and Software Pipelining**
- Local Instruction Scheduling; List-scheduling algorithm
- Architectural complications
- Beyond basic-blocks: hyper-blocks and trace scheduling
- Software Pipelining

**Lecture 5: Software Prefetching**
- Basic concepts in prefetching
- When to prefetch? Prefetch distance
- Case Study: Stencil codes and the use of data dependence analysis

**Lecture 6: Loop Transformations for Vectorization and Parallelization**
- Basic Vectorization and Parallel Execution Concepts
- Vectorization and Parallelization Tests
- Examples

**Lecture 7: Message Passing and Distributed Shared Memory Machines**
- TBD
- TBD
- TBD

**Lecture 8: Empirical-guided Optimization and Tuning**
- TBD
- TBD
- TBD

**Lecture 9: Compiling for (re)configurable Hardware**
- TBD
- TBD
- TBD
Projects Topics: Below is a tentative description of projects topics.

Topic 1. Analysis and Transformations for Caches and Registers
In this project you are asked to study and data dependences of a real code (focus on a specific kernel) and evaluate the impact of several of the transformations (most notably loop transformations) studied in this class. You are expected to outline the basic analysis and possible trade-offs and evaluate several of the transformation parameters on an high-end machine. Your infrastructure can be either a single processor or a multi-processor architecture.

Topic 2. Empirical Evaluation of Compiler Transformations
In this project you are asked to evaluate the efficacy and interplay between existing compiler transformations in the context of a sequential code. One possible outcome is to explore a wide range of compiler switches (such as the gcc flags) for a specific set of codes and develop a model for which optimizations are best suited for each type of kernel code. The evaluation should be done in the context of a sequential code.

Topic 3. Analysis of a Parallel Application Code
In this project you are asked to use existing performance monitoring tools such as the HPC Toolkit and instrumentation packages to evaluate the performance and potential benefits of parallel execution. An analysis of grain size, synchronization and load balancing is expected along with the possible use of transformations studied in this class to ameliorate the performance issues you have uncovered.

Topic 4. Study of Software Resiliency Techniques
In this project you are asked to develop a set of program transformation techniques, alongside their required analysis for dealing with transient architecture errors. You’ll have to make several assumptions about the architecture interface as well as about the high-level programming language interface. This is clearly a more research oriented and analytical project for which you are expected to performance some amount of state-of-the-art research literature and develop some techniques on your own.