Course Number: 3460:641
Course Name: Optimization for Parallel Compilers
Course Credits: 3
Schedule: On demand

Syllabus Date: June 9 2008
Prepared By: Tim O’Neil

Prerequisites:
Graduate standing and knowledge of C.

Text:

Bulletin Description:
Advanced analysis and transformation strategies to support automatic vectorization and parallelization of code, emphasizing restructuring to improve instruction scheduling.

Detailed Description:
See “Bulletin Description”. We will also apply these strategies to the problem of optimizing for advanced microprocessors.

Course Goals:
To introduce the theory of dependence analysis and code optimization. It will review of the practice of compiler construction, with a major course component being the design and implementation of a compiler-like optimizer.

Topics:
1. Compiler Challenges for High-Performance Architectures
   a. Parallel and vector architectures
   b. The problem of parallel programming
   c. Bernstein's conditions and the role of dependence
   d. Compilation for parallel machines and automatic detection of parallelism
2. Dependence Theory and Practice
   a. Fundamentals, types of dependences
   b. Construction of direction and distance vectors
3. Parallelization and Vectorization
4. Dependence Testing
   a. Separable dependences
   b. The gcd and Banerjee tests
   c. Exact dependence testing
5. Transformations
   a. Loop normalization
   b. Scalar data flow analysis
   c. Induction variable substitution
d. Scalar renaming

   a. Loop distribution and its safety
   b. The Kuck vectorization principle
   c. The layered vector code-generation algorithm and its complexity
   d. Loop interchange
   e. Loop Interchange
   f. Loop Skewing
   g. Scalar and array expansion
   h. Forward substitution
   i. Alignment
   j. Code replication
   k. Array renaming
   l. Node splitting
   m. Pattern recognition
   n. Threshold analysis
   o. Symbolic dependence tests
   p. Parallel code generation and its problems

**Computer Usage:**
The construction of an optimizer which makes a source-to-source conversion of programs written in a subset of the C language.

**References:**