15-745 Semester Project

We prefer that projects are done in groups of two, although groups of three may be permitted depending on the scale of the project (ask the instructor for permission before forming a group of three). If you are looking for a partner, you can use 15-745 Piazza.

Proposal

Due 3/18. The project proposal should be 3 pages; single spaced, one or two columns, 10 point font or larger. Describe the project idea/application, what work must be done (suggesting how it can be partitioned among you) and what resources you will need (including software systems you already have access to). Concentrate on convincing us that it will pertain to the course, that you will be able to complete it, and that we will be able to evaluate it. The third page should be dedicated to providing an outline of your intended final paper, identifying the specific experiments to be run and what questions they will answer.

In your proposal, provide three types of goals: 75% goals, 100% goals, and 125% goals. Think of these as the equivalent of a B grade, an A grade, and a "wow!" grade. The goals can be dependent or independent of the prior goals.

Think also about providing answers to the Heilmeier Questions in your proposal, as appropriate.

Project Milestone Report

Due 4/11. The milestone report should be at most 3 pages. It should discuss the following points:

- A summary of the motivation for the project, and how it compares with related work. Has anything changed based upon feedback from your proposal or as a result of your research so far?
- A re-iteration of your proposed goals, what progress you have made to date on those goals, and what your timeline is for accomplishing the rest of them by the end of the semester.
- A list of the major components of your evaluation plan. What's been completed and what did you learn?
- What are the current concerns with this project and its progress? Any stumbling blocks discovered?

Poster Presentation

Held 4/29 during class time. Rather than having oral presentations to present your projects to the class, we will instead have a poster session on the last day of class. We will provide easels, posterboard, and tape. You should bring up to eight 8.5x11" pages to tape onto the posterboard when you first arrive.

The advantage of a poster sesson is that it is interactive: you can ask other people questions one-on-one, and you can prioritize your time based upon which projects interest you the most. One member of your group should stand near your poster while the other group member(s) circulates to ask people about their posters. You will take turns doing this, so everyone will spend part of the time presenting their poster, and part of the time asking others about their posters.

The poster session is open to the public. Please feel free to invite others to attend it.

Final Project Report

Deadline extended: Now due 5/3. This report should be written in the format of a research paper. For further details, see final_report.pdf.

In addition to the report, please submit **a tar file of your source code** to Blackboard.

Here are two example final project reports from last year:

- Optimizing Matrix Operations Using Novel DRAM Access Primitives
- Data Layout Optimzation for GPU Programs

Prior Project Topics (past 4 years)

Architectural Primitives and Hardware Support:

- An Evaluation of a Block-Structured ISA. Chris Fallin and Gennady Pekhimenko
- Compiler Support for Hardware Compression. Yixin Luo, Hongyi Xin, and Nandita Vijaykumar
- High Level Synthesis for CoRAM. Gabriel Weisz

Approximate Computing:

- Approximating Functions for Grid Domains. Danny Zhu, Yuzi Nakamura, and Ben Humberston
- Just-In-Time Approximate Computing. Joseph Lee and Naman Jain

Code Optimization and Scheduling:

- A classifier for ordering and selection of dependent LLVM code passes. Brendan Meeder, Jamie Morgenstern, and Richard Peng
- Aggressive Scheduling for Numerical Programs. Richard Veras, Flavio Cruz, and Berkin Akin
- Clustering for optimizing compiler optimization parameters. Siddhartha Jain and Christian Kroer
- Implementing Profile-Guided Speculative Code Motion in LLVM. Ben Jaiyen and Jamie Liu
- Learning Function Inlining. Yair Movshovitz-Attias and Dana Movshovitz-Attias
- Phase Order Search with Feature Selection. Joao Martins and David Henriques
- Predicting Sequence of Parameterized Transform Passes using Machine Learning Techniques. Ashique Khudabukhsh and Jayant Krishnamurthy
- Predicting Unroll Factors Using Supervised Classification. Nan Li and Julian Shun
- Using Natural Language Hints to Improve Scheduling and Prefetching in JAVA. Alejandro Carbonara, Wenlu Hu, and Jiyuan Zhang

Concurrency, Parallelism, and Vectorization:

- Automatic Loop Vectorization in Javascript Using SIMD.js. Tom Chittenden and Mario Dehesa Azuara
- Automatic Multi Threaded Code Generation for Signal Processing Transforms. Thom Popovici and Darya Kurilova
- Automatic Parallelism in Octopus. Shiva Kaul and Kristina Sojakova
- Architecture-Specific Compiler Optimizations for Exploiting SIMD-Level Parallelism in Programs. Nipunn Koorapati and Parag Dixit
- Compiler Optimizations Using Data-Triggered Threads. Alex Limpaecher and Deby Katz
- Concurrent Speculative Optimization Framework for Just-In-Time Compiler. Andrew Kun-Diat Tan, Yue Xing, and Pratch Piyawongwisal
- CREAT: Computational Reduction of Expressions using Algebraic Techniques. Bao Hong Tan and Vincent Teo
- Eliminating Loop Iterations using a Polyhedral Model. Rachata Ausavarungnirun and Donghyuk Lee

- Enabling Loop Parallelization with Decoupled Software Pipelining in LLVM. Ameya Velingker and Dougal J. Sutherland
- High Performance Pipeline Compiler. Yong He and Yan Gu
- Implementation of Decoupled Software Pipelining (DSWP) in the LLVM Compiler Infrastructure. Fuyao Zhao and Mark Hahnenberg
- Mapping Parallelism to Multi-cores. Mehdi Samadi and Ankit Sharma
- Multicore Prefetching with Helper Threads. Sinziana Munteanu and Richard Wang
- Vectorization of Programs in CPS Form. Anton Bachin and Anand Subramanian

Domain-Specific Optimizations:

- An Optimizing Compiler for a 3D Rendering DSL. Jesse Dunietz and Nicolas Feltman
- Finding Optimization Opportunities in Android Inter-Component Communications. Waqar Ahmad, Mauricio Soto, and Chu-Pan Wong
- Partial Redundancy Elimination in CMORT. Luke Zarko
- Range Analysis for IonMonkey. Ryan Pearl and Michael Sullivan
- Setty: Optimizing a Set Language. Adam Blank and Andrey Kostov

Memory Optimizations:

- Graph-Based GPU Coalescing Optimization. Yu Wang and Guanglin Xu
- Improving Cache Locality with Coalesced Map-Reduce Operations. Michael Choquette and Rokhini Prabhu
- Machine Learning for Instruction Prefetching. Sarah Allen and Brandon Amos
- Memory-Conscious Data Layout. Justin Meza and Yoongu Kim
- Modeling memory usage patterns of GPU programs written for the NVIDIA GPU architecture. Cyrus Omar and Salil Joshi
- Optimizing Applications that Traverse Irregular Data Structures through SIMD with Prefetching. Dong Zhou and Nathan Fulton
- Optimizing Array Locality via Memory Layout Reorganization. Junchen Jiang and Xuezhi Wang
- Optimizing Matrix Operations Using Novel DRAM Access Primitives. Joy Arulraj, Anuj Kalia, and Jun Woo Park
- Restructuring Memory Access for Optimal Cache Performance. Prashanth Suresh, Zhe Qian, and Adu Bhandaru
- Software Prefetching in LLVM. Nathan Snyder and Q Hong
- Virtual Views. Kevin Chang and Vivek Seshadri

Profile-Guided Optimization and Specialization:

- Dynamic Just-In-Time Value Specialization. Alejandro Carbonara and Aram Ebtekar
- Instrumenting V8 to Measure the Efficacy of Dynamic Optimizations on Production Code. Michael Maas and Ilari Shafer
- Profile Guided Memoization For Purely Functional Languages. Shen Chen Xu and Jakub W. Pachocki

Register Allocation:

- Improved Heuristics for Coalescing. Carol Wang and Patrick Xia
- Interprocedural Register Allocation for LLVM. Benjamin Blum and Nicholas Tan
- Profile Guided Register Allocation and Scheduling. Paul Sastrasinh and Robert Shih
- Towards a Faster Register Allocator via Matrix Multiplicatives Weights. David Witmer and John Wright

Static Analysis:

- Applying Compiler Optimizations on Symbolic Execution. Thanassis Avgerinos and Alexandre Rebert
- Binary Inference on Typed Assembly Language. Carlo Angiuli and Matthew Maurer
- Combinatorial Algorithms for Specification Inference in Information Flow Problems. Sarah Loos and Akshay Krishnamurthy
- Data Reachability Analysis for Partial Compile-Time Garbage Collection in Java. Samir Jindel and Logan Brooks
- If-Conversion to Reduce Flow-Based Side-Channel Leaks. Jonathan Burket and Samantha Gottlieb
- Immutability Ranges in Object-Oriented Programming Languages. Michael Coblenz and Colin White
- Interprocedural Variable Liveness Analysis for Function Signature Recovery. Miguel Araujo and Ahmed Bougacha
- Memory Safety Bug Hunting with BAP. Tiffany Bao, Dominic Chen, and Rijnard van Tonder
- Scalable context-sensitive and flow-sensitive alias analysis. Hugues Bruant
- Static Information Analysis and Machine Learning Analysis on Automated Detecting Browser Extension Vulnerabilities. Jaicheng Hong, Boyan Li, and Qinsi Wang
- Symbolic Execution in Difficult Environments. David Renshaw and Soonho Kong

Project Management

You should strongly consider using either Subversion or Git to perform source code control for your project and the paper you write describing it.

I also strongly suggest writing your course project report using LaTeX. It is the de-facto tool in which most CS research papers are written. While it has a bit of start up cost, it's *much* easier to collaboratively write complex research papers using LaTeX than Word.

Writing Papers

- An Evaluation of the Ninth SOSP Submissions - or - How (and How Not) to Write a Good Systems Paper by Roy Levin and David D. Redell
- Collected resources about writing and research