CS 8803 Dynamic Compilation

Instructor
Prof. Nate Clark
Office: 2340 Klaus
Office hours: Tues. 2-3 or by appointment

Website
T-Square

Schedule
TTh 3:05 – 4:25 IC 217

Description
We will look at the design, challenges, and opportunities of dynamic compilation and virtual machines, primarily from the perspective of program optimization and architecture impact. Topics include interpretation and translation implementation techniques, case studies of actual dynamic compilers in use (e.g., Java VMs, Microsoft's Common Language Interface, and Adobe's Flash Player), and architectural support to enhance dynamic translation. We will cover several recent research papers on these topics and students will be expected to complete a significant course project.

Recommended Text
"Virtual Machines: Versatile Platforms for Systems and Processes" by Smith and Nair

This book is a very good reference, but most of the material in this course will come from reading recent research papers.

Prerequisites
At least one course on computer architecture (including caching, branch prediction, and virtual memory), and one course on compilers. Basic rule of thumb: if you don't know what a TLB and a Basic Block are, don't take this class. If you know what both of them are and how they work, you'll probably be fine.

Grading
10% - Participation
15% - Presentation(s)
8% - Paper writeup
2% - Project proposal
5% - Status report 1
5% - Status report 2
30% - Final project
25% - Exam

>= 90% is an A, >= 80% is a B, …
I reserve the right to decrease those numbers (i.e., curve the course).

Presentations (text gratuitously borrowed from Prof. Amer Diwan)

In most classes one student will be responsible for presenting the key ideas in the papers assigned to that class. The presenter should schedule an appointment with me to discuss the assigned papers at least two days before the class. The presenter should of course, carefully and thoroughly read the papers for the class. In addition, if necessary, the presenter is responsible for reading other papers that may shed some light on the assigned papers. If there is more than one papers assigned on a particular day then the presenter must give a single talk that integrates the ideas of all the papers. *All students are required to read the assigned papers before the class.*

A presentation must contain the following:

1. A summary of the ideas in the assigned papers. Note that since everyone will have read the papers, the presenter need not present all the elementary ideas in the paper, just the main concepts. However, she/he should be prepared to discuss details of the paper in response to questions by the students or the instructor. Unless the paper is a very difficult one, you should plan to spend no more than half the class on this part of the presentation (if you are in doubt about this, it is a good thing to discuss with the instructor in your one-to-one meeting).
2. Discussion of any additional papers that you read on the topic. Since other students will not have read these papers you need to be more pedantic about these papers.
3. A discussion of how the ideas in the paper fit in with the other papers we have discussed in the term
4. A critique of the paper. Do the papers present useful ideas? Is the research in the papers thorough? Was the writing clear? Was the evaluation in the paper (if any) convincing? Do the ideas work for real programs on real machines?
5. Any interesting research ideas that you got from reading the paper. If the paper already has a future works section then you may discuss interesting ideas there as well.
6. Are there other areas where you can imagine using the ideas in the paper?
7. Questions to prompt further discussion in class

The bottom line is that I want you to spend more of your presentation analyzing the ideas in the paper rather than simply presenting the ideas.

The presenter has seven days after the presentation to submit a write-up on the paper. These write-ups will be available from the class web page for the benefit of all the students.

Guidelines for giving a talk

Like all courses, this course too relies heavily on the quality of presentations and class participation. A badly given presentation can render even the simplest topic incomprehensible. A good presentation can inspire! Here are some general guidelines that I would like you to follow:

0. Your talk should last about 40 minutes, which will leave time for discussion.
1. Minimize the amount of text on a slide. A slide should typically contain just a few bullets (3-5) or a figure/table and a sentence or two. Each bullet should be short--you should strive to contain each bullet to a line. A slide is not the place to write out long definitions, complex formulas, long algorithms, etc. If you want to describe an algorithm, use the slides to get the intuition across (hopefully using figures). Be prepared to answer detailed questions however. If the paper contains complex algorithms, it may be

helpful to have optional slides containing the algorithm. These slides would be used only in response to questions requiring the details.

2. Use figures instead of text when possible. A figure may not be as unambiguous or precise as a mathematical formula or an algorithm, but that is fine: more details can always be brought up in the discussion period.

3. Allocate 2-3 minutes for a slide. For a 1 hour presentation, it means no more than 30 slides.

4. Use large font sizes (18 points or bigger).

5. Use colors judiciously. Some colors, such as green and yellow are not too visible on an overhead.

6. Make an outline of a talk before you start preparing the slides. It will help you prepare a talk that feels smooth and continuous rather than fragmented.

7. For each slide try to think of a one-liner take home message from that slide. If there are more than one "one-liners" then maybe you need to break your slide into multiple slides.

Tentative Schedule

Date	**What are we talking about**	**Who's Talking / Other Notes**
Jan 12	Intro / What is a DC? How are DCs organized?	Nate
Jan 14	Interpretation techniques - Smith and Nair Chapter 2.1 – 2.4	Nate
Jan 19	Translation techniques/code caching - Smith and Nair Chapter 2.5 – 3.11	Nate
Jan 21	Continuing the previous 2 lectures	Nate
Jan 26	Selecting fragment size/fragment linking - Trace Fragment Selection within Method-based JVMs by Merrill - Evaluating Indirect Branch Handling Mechanisms in Software Dynamic Translation Systems – Hiser	
Jan 28	Code caching - Managing Bounded Code Caches in Dynamic Binary Optimization Systems – Hazelwood and Smith - Process-Shared and Persistent Code Caches - Bruening	
Feb 2	ISA-specific challenges (precise exceptions, eflags, etc.) - Precise Exception Semantics in Dynamic Compilation – Gschwind	
Feb 4	Other DC overheads and solutions (shadow page tables, IO, etc.) - A Comparison of Software and Hardware Techniques for x86 Virtualization – Adams - Intel Virtualization Technology: Hardware support for efficient processor virtualization by Neiger	
Feb 9	Overheads II - Reducing the overhead of dynamic compilation – Krintz - Reducing Startup Time in Co-Designed Virtual Machines – Hu	
Feb 11	Project Day	Nate

Feb 16	Online escape and alias analysis - Fast online pointer analysis – Hirzel	
Feb 18	Garbage Collection - Uniprocessor Garbage Collection Techniques – Wilson	Project proposal due
Feb 23	Support for streaming programming models - The Stream Virtual Machine – Labonte - Flextream: Adaptive Compilation of Streaming Applications for Heterogeneous Architectures – Hormati	
Feb 25	Handling Dynamic Data Types - "Trace-based Just-in-Time Type Specialization for Dynamic Languages" by Gal, Eich, etc.	
Mar 2	Online Profiling I - A Framework for Reducing the Cost of Instrumented Code – Arnold - Software profiling for hot path prediction: Less is more. – Deusterwald	
Mar 4	Online Profiling II - Branch-on-Random – Lee - Online Phase Detection Algorithms – Nagpurkar	Status report 1 due
Mar 9	Optimizations I (Power, Precomputation) - A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance – Wu - Accelerating and Adapting Precomputation Threads for Efficient Prefetching – Zhang	
Mar 11	Optimizations II (Code specialization) - Runtime Specialization With Optimistic Heap Analysis – Shankar - Hardware Atomicity for Reliable Software Speculation – Neelakantam	
Mar 16	Optimization III (Parallelization) - Jrpm System for Dynamically Parallelizing Java Programs by Chen - Thread Tailor: Dynamically Weaving Threads Together for Efficient, Adaptive Parallel Applications - Hu	
Mar 18	Applications to RAS - When Virtual is Better than Real – Chen - Unmodified Device Driver Reuse and Improved System Dependability via Virtual Machines – LeVasseur	
Mar 30	Applications to security (taint flow analysis, input sanitization) - Secure and Practical Defense Against Code-injection Attacks using Software Dynamic Translation by Hu - Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software by Newsome	Status report 2 due

Apr 1	Applications to software engineering (invariant enforcement, developer feedback) - DITTO: Automatic Incrementalization of Data Structure Invariant Checks (in Java) – Shankar - A Practical Approach to Exploiting Coarse-Grained Pipeline Parallelism in C Programs – Thies	
Apr 6	Case study: Google V8 - Watch this 1 hour video: http://channel9.msdn.com/shows/Going+Deep/Expert-to-Expert-Erik-Meijer-and-Lars-Bak-Inside-V8-A-Javascript-Virtual-Machine/	
Apr 8	Case Study: Microsoft CLR	
Apr 13	Case study: IBM's Daisy/BOA - Dynamic binary translation and optimization – Ebcioglu	
Apr 15	Case study: Transmeta Crusoe - The Technology Behind Crusoe Processors - Klaiber	
Apr 20	No class. Go to IPDPS at the Atlanta Sheraton or finish up your projects	
Apr 22	No class. Go to IPDPS at the Atlanta Sheraton or finish up your projects	Final Project due before midnight on the 23rd
Apr 27	Case Study: Google Native Client - Native Client: A Sandbox for Portable, Untrusted x86 Native Code – Yee	
Apr 29	Case study: Google Dalvik - http://www.youtube.com/watch?v=ptjedOZEXPM	
	Final Exam	