During the last two years of my undergraduate career at the University of Texas, I was a part of the Computer Architecture and Technology Laboratory (CART) group led by Professors Doug Burger and Steve Keckler. My research, supervised by Professor Burger, focused specifically on memory disambiguation, a significant challenge in modern microprocessor design.

Background & Motivation: Microprocessor performance has been increasing exponentially for the past 30 years, one of the key enabling technologies that has powered the computing revolution. Unfortunately, continued improvement at historical rates is far from guaranteed, since many technical challenges stand in the way of future progress. One of these challenges is memory disambiguation. Modern processors have many operations in flight at once, including memory reads and writes, which can be executed out of order to improve performance. The fact that it is unknown in advance which reads and writes will access the same memory location poses a problem for the effective reordering of operations. If a load reads from memory before an earlier store writes to the same location, the load may read an incorrect value and the processor's pipeline must be flushed, resulting in performance degradation. Memory disambiguation was an enormous problem for dataflow architectures in the 1970s/80s, which required that all instruction dependences be known statically; this forced dataflow machines to use unconventional (side-effect free) languages, which surely hindered their adoption. Until now, memory disambiguation has been one of the primary impediments to scaling the performance of high-performance, single-threaded microprocessors.

Project 1: Unordered Load-Store Queues: My contributions to this area have consisted of two parts. In the fall of 2006, I collaborated with then-graduate student Simha Sethumadhavan on a project for which I did the Verilog implementation and verification of an unordered Load-Store Queue (LSQ) design. LSQs buffer load and store instructions until they commit, enable direct forwarding of data from stores to loads, and detect if a dependent load has executed too early. Our work shows that by allocating LSQ slots to instructions when they issue rather than when they dispatch ("late binding"), the area and power efficiency of LSQs can be improved. This design requires the LSQ structure to be unordered (entries in the LSQ are not physically ordered according to their age), but my Verilog implementation shows that it is trivial to adapt LSQ functions for the unordered structure.

Project 2: Dependence Prediction: From this first project, I gained a deeper understanding of the research process and the low-level details of computer architecture. This allowed me, during my senior year, to take on a project at a higher level of abstraction setting out to solve the larger problem of memory ordering. Modern processors rely on memory dependence prediction to execute load instructions as early as possible: they predict which reads from memory should wait for a matching write instruction and which reads can safely be issued early. Dependence predictors use a load's previous execution history to make predictions intended to minimize performance losses due to loads executing too early (and therefore causing flushes of the pipeline) or to loads being held back longer than necessary (and thus losing valuable opportunities for parallelism).

Previous Dependence Predictors: The early work on dependence predictors began with Moshovos and Sohi's PC-matching predictor [1] and with the Alpha 21264 load-wait table [2]. Chrysos and Emer's Store Sets predictor [3] achieves close to ideal performance, defined as each load waiting only for the exact stores, if any, that will forward values to the load.

However, previous dependence predictors were shown to be near-ideal under several key assumptions that have since changed. Global wire delays have resulted in the emergence of partitioned architectures, such as CMPs and tiled architectures. Distributed architectures may

execute single-threaded code without a single centralized fetch and/or execution stream. This will make it challenging to deploy predictors such as Store Sets, which require observation of the stream of fetched instructions to synchronize loads with specific stores. Previously proposed dependence prediction mechanisms also rely on global execution information to track the completion of stores that trigger the wakeup of deferred loads. These factors result in the need for new predictors that work effectively for large-window distributed microarchitectures.

My Counting Dependence Predictors: In my research, I proposed a class of dependence predictor designs that we call *Counting Dependence Predictors*. CDPs are designed to work well in distributed architectures in which accurate predictions must be made with as little remote information as possible. The enabling feature in CDPs is that they predict only *local* events for which a particular dynamic load should wait (and can thus be oblivious to the *global* fetch stream). In my implementation of a CDP, a table of counters indexed by a load's PC will indicate the load's wakeup event. This event is simply a number of matching stores for which a load must wait: zero (aggressive), N (already arrived or arriving later), or all of them (conservative). N is 1 in my implementation. Because *any* earlier store to the same address is considered a match for a load, rather than some *specific* fetched store (as Store Sets requires), the predictor mechanism is decoupled from reliance on the fetch stream. If deferred loads are held at their cache bank, information about matching stores will be available locally.

Evaluation & Results: I evaluated CDPs in the context of the TFlex microarchitecture [4], a fully distributed tiled architecture that supports an issue width of up to 64 instructions and an execution window of up to 4,096 instructions. Since control decisions, instruction issue, and dependence prediction may all occur on different tiles, a distributed protocol for handling efficient dependence prediction is needed. Though my work describes a protocol implementation specific to TFlex, the ideas behind CDPs are applicable to any architecture with distributed fetch and distributed memory banks.

The best-performing CDP configuration in my work achieves 92% of the performance of perfect memory disambiguation. Idealized, centralized implementations of Store Sets and Yoaz et al.'s Exclusive Collision Predictor [5] achieve 97% and 94% of perfect, respectively. Of these and other predictors studied, only the load-wait table is straightforward to implement in a distributed fashion, as it is essentially a degenerate CDP. The CDP I evaluate outperforms Load-Wait by 11%.

Publications: This work appeared in the 35th International Symposium on Computer Architecture (*Counting Dependence Predictors*, 2008); I presented it there in Beijing, China in June 2008. This work also culminated in my undergraduate honors thesis for the Turing Scholars Honors Program in Computer Sciences. My work with Simha appeared in the 34th International Symposium on Computer Architecture (*Late-Binding: Enabling Unordered LSQs*, 2007).

References:

[1] A. Moshovos and G. S. Sohi. Dynamic speculation and synchronization of data dependences. In *24th International Symposium on Computer Architecture*, 1997.

[2] Compaq Computer Corporation. *Alpha 21264 Microprocessor Hardware Ref. Manual*, 1999.

[3] G. Z. Chrysos and J. S. Emer. Memory dependence prediction using store sets. In *25th Annual International Symposium on Computer Architecture*, 1998.

[4] C. Kim, S. Sethumadhavan, *et al.* Composable lightweight processors. In *40th International Symposium on Microarchitecture*, 2007.

[5] A. Yoaz, M. Erez, R. Ronen, and S. Jourdan. Speculation techniques for improving load related instruction scheduling. In *26th International Symposium on Computer Architecture*, 1999.