CS 701

Charles N. Fischer

Fall 2001

http://www.cs.wisc.edu/~fischer/cs701.html

Class Meets

Tuesdays & Thursdays, 9:30 — 10:45
1263 Computer Sciences

Instructor

Charles N. Fischer
5397 Computer Sciences
Telephone:    262-6635
E-mail:       fischer@cs.wisc.edu
Office Hours:
              10:30 - Noon, Mondays &
              Wednesdays, or by appointment

Teaching Assistant

James Nugent
1308 Computer Sciences
Telephone:    262-6602
E-mail:       damion@cs.wisc.edu
Office Hours:
              1:30 - 2:30 Tuesdays,
              10:30 - 11:30 Fridays,
              or by appointment

Key Dates

- September 25:    Project 1 due
- October 23:     Project 2 due (tentative)
- October 30:     Midterm (tentative)
- November 20:    Project 3 due (tentative)
- December 13:    Project 4 due
- December ??:    Final Exam, date to be
determined
Class Text

There is no required text. Handouts and Web-based reading will be used.

Suggested reference:


Instructional Computers

Departmental SPARC Processors (nova1-nova60)
You may use your own workstation if it is has a SPARC processor (test using dmesg|grep cpu)
Otherwise log onto a SPARC processor to do SPARC-specific assignments

CS701 Projects

1. SPARC Code Optimization
2. Global Register Allocation (using Graph Coloring)
3. Global Code Optimizations
4. Individual Research Topics

Academic Misconduct Policy

- You must do your assignments—no copying or sharing of solutions
- You may discuss general concepts and ideas
- All cases of Misconduct must be reported.
- Penalties may be severe.
Reading Assignment

- Get Handout #1 (SPARC Architecture Information) and Handout #2 (Chapter 15, Code Optimization) from DoIt.
- Review Handout #1 and read section 15.2 of Chapter 15.
- Read Assignment #1

Overview of Course Topics

1. Register Allocation
   
   Local Allocation
   Avoid unnecessary loads and stores within a basic block. Remember and reuse register contents. Consider effects of aliasing.

   Global Allocation
   Allocate registers within a single subprogram. Choose “most profitable” values. Map several values to the same register.

   Interprocedural Allocation
   Avoid saves and restores across calls. Share globals in registers.

2. Code Scheduling
   
   We can reorder code to reduce latencies and to maximize ILP (Instruction Level Parallelism). We must respect data dependencies and control dependencies.

   \[
   \begin{align*}
   &ld \ [a], \%r1 \quad ld \ [a], \%r1 \\
   &add \ %r1, 1, \%r2 \quad mov \ 3, \%r3 \\
   &mov \ 3, \%r3 \quad add \ %r1, 1, \%r2 \\
   \end{align*}
   \]
   (before) \quad (after)

3. Automatic Instruction Selection
   
   How do we map an IR (Intermediate Representation) into Machine Instructions?
   Can we guarantee the best instruction sequence?

   Idea—Match instruction patterns (represented as trees) against an IR that is a low-level tree. Each match is a generated instruction; the best overall match is the best instruction sequence.
Example:
\[ a = b + c + 1; \]
In IR tree form:

\[
\frac{s}{\begin{array}{c}
  \frac{a_{adr}}{+} \\
  \frac{\downarrow}{+} \\
  \frac{\uparrow}{l}
\end{array}}
\]

Generated code:

\[
\begin{align*}
  &ld \ [\%fp+b_{offset}],%r1 \\
  &ld \ [c_{adr}],%r2 \\
  &add \ %r1,%r2,%r3 \\
  &add \ %r3,1,%r4 \\
  &st \ %r4,[a_{adr}]
\end{align*}
\]

Why use four different registers?

4. Peephole Optimization
Inspect generated code sequences and replace pairs/triples/tuples with better alternatives.

\[
\begin{align*}
  &ld \ [a],%r1 \quad ld \ [a],%r1 \\
  &mov \ const,%r2 \quad add \ %r1,\const,%r3 \\
  &add \ %r1,%r2,%r3 \\
  (before) \quad (after)
\end{align*}
\]

\[
\begin{align*}
  &mov \ 0,%r1 \quad OP \ %g0,%r2,%r3 \\
  &OP \ %r1,%r2,%r3 \\
  (before) \quad (after)
\end{align*}
\]

But why not just generate the better code sequence to begin with?

5. Cache Improvements
We want to access data & instructions from the L1 cache whenever possible; misses into the L2 cache (or memory) are expensive!

We will layout data and program code with consideration of cache sizes and access properties.

6. Local & Global Optimizations
Identify unneeded or redundant code.
Decide where to place code.
Worry about debugging issues (how reliable are current values and source line numbers after optimization?)