Optimizing Compilers Projects in EDAN70 2018

- One or two persons per group
- Two (but not three) groups can do the same project
- Select a project on Friday 9/11 the latest

Optimizing Compilers Projects in EDAN70 2018

- Project with Kenny Ranerup from Packet Architects
  `kenny.ranerup (at) packetarc.com`
- Projects with Karl Hylén from ARM
  `karl.hylen (at) arm.com`
- Iterated register coalescing in LLVM
- Minimum-cut speculative SSAPRE in vcc
- Boassinot out from SSA in vcc or LLVM
- Linear-scan register allocation on SSA form in vcc or LLVM
- Coloring-based coalescing in vcc
- Dominance-based code duplication in vcc or LLVM
- Own suggestion

Iterated register coalescing in LLVM

- LLVM has four register allocators:
  - Fast — at basic block level
  - Basic — allocates to small live ranges first
  - Greedy — allocates to large live ranges first (may reverse earlier decisions)
  - PBQP (partitioned boolean quadratic programming, developed for irregular architectures). There is an optimal solver for this but a heuristic is used.
- Implement the algorithm in the book, for a subset of the data types (otherwise likely too much work)
- Possible continuation: exjobb in LLVM

Minimum-cut speculative SSAPRE in vcc

- Implement SSAPRE and extend it with MC-SSAPRE algorithm
- Use statistics from vcc for edge costs
- Compute a minimum cut (Ford-Fulkerson from EDAF05) and insert the expressions at the cut
- Measure performance effects
- Possible continuation: exjobb in LLVM or GCC

Boassinot out from SSA in vcc or LLVM

- Coalescing based on fast live analysis using dominance
- Implement and measure:
  - Performance effects on compiled code
  - Compile-time overhead with standard from SSA algorithm (the one in vcc)
- Possible continuation: exjobb in LLVM or GCC

Linear-scan register allocation on SSA form in vcc or LLVM

- The client Java HotSpot VM uses linear scan
- With SSA form the interference graph is chordal (for every cycle with at least four nodes there is an edge not in the cycle which connects two nodes in the cycle)
- Based on paper from Irvine and Oracle Labs in Austria
- Implement and measure:
  - Performance effects on compiled code
  - Compile-time overhead
- Possible continuation: exjobb in LLVM or GCC

Coloring-based coalescing in vcc

- First do Iterated register coalescing
- Recall: at pop of u in Chaitin with this algorithm from IBM Tokyo:
  1. try a real color used by a move-related node v, or
  2. try a real color, or
  3. try an extended color used by a move-related node v, or
  4. create a new extended color.
- Implement and measure:
  - Performance effects on compiled code
  - Compile-time overhead
- Possible continuation: exjobb in LLVM or GCC

Dominance-based code duplication in vcc or LLVM

```
if (x > 0)
        y = x;
else
        y = 0;
return y + 2;
```

```
if (x > 0)
        return x + 2;
else
        return 2;
```

- Always duplicating is a bad idea
- Using the dominator tree the effect of duplication is simulated and analysed to decide whether to duplicate or not
- Compile-time and performance effects
- Possible continuation: exjobb in LLVM or GCC

LLVM Superoptimizer

- Add support for floating point instructions in the LLVM superoptimizer
- See: https://github.com/google/souper
- This can become an exjobb in LLVM during the spring or later

LLVM Optimization Passes Ordering

- Make a tool with which the best order of optimization passes can be found for a particular input file (since obviously different input files may benefit from different orders).
- This tool can be quite general and not necessarily made for a particular CPU architecture
- This can become an exjobb during the spring or later