HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Reconfigurable & embedded
Digital Systems

Stéphane Donnet
Rick Wertenbroek

HPC - 2016

Super Optimizers

Focusing on LLVM

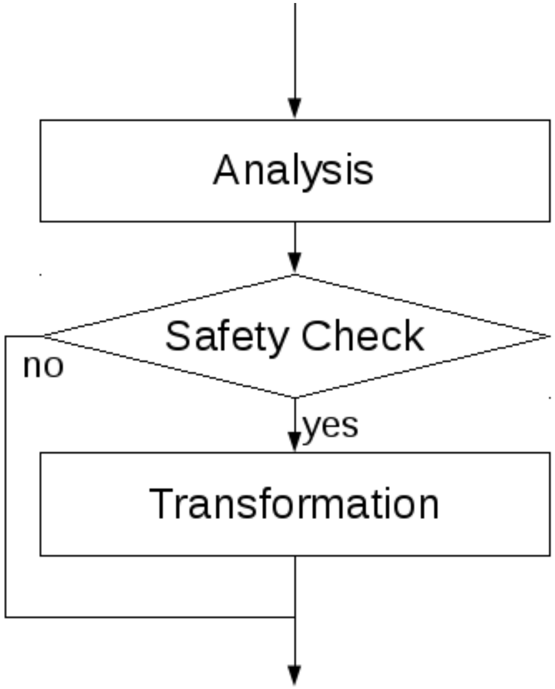
Summary

- Classic Optimizations
- Super Optimizers
- SMT (Satisfiability Modulo Theories)
- Souper
- Who are Super Optimizers for ?
- Which optimizations are worth implementing ?
- Demo, Conclusion and Questions

HPC
2016

Classic Optimizations

- As done in GCC or LLVM with -OX



What are Super Optimizers

- Basically they are programs that search and discover optimizations.
- There are two main advantages
    - We don't have to code the optimizations by hand
    - They discover optimizations we never thought of
- Modern Super Optimizers prove their optimization is correct before using it. (Real proof, not just some test cases)

Super Optimizers

- First appeared in 1987 – Massalin ’87
  - Brute-Force the smallest machine code for a given task, verified with just a few tests.
- Super Optimizer to find new optimisations
  - Lots of easy optimizations found to implement in existing compilers – Sands ’11
- From unreliable to reliable
  - Many optimizations where changed the behaviour of the code. Solution : add automated theorem prover – Denali ‘01 ‘03

SMT (Satisfiability Modulo Theories)

- Instruction are converted into predicates
- Thoses predicates form a theorem
- The solver tries to prove this theorem with less predicates (other predicates may be used)
- This « proof » is converted into instructions again
- We are sure these instructions do the same thing

Souper

https://github.com/google/souper

- Open Source Super Optimizer, works with LLVM
- Works with an Intermediate Representations (bytecode, basically a simplified LLVM IR)
- Transforms instructions in a DAG
  - Easy to (de)serialize, easy to cache
- Turns instructions into predicates (SMT-LIB format)
  - Works with a SMT Solver to prove the optimizations
- Passes through LLVM again until no more changes occur. (will remove dead code etc…)

Who are Super Optimizers for ?

- Compiler users (of course)
- LLVM Devs (to implement the missing optimizations)
- To find bugs in Compilers (Found bugs in GCC, LLVM and even CompCert)

- Everybody benefits !

Which optimizations are worth implementing ?

- How to answer ?
  - Profiling of course !

- 1) The most used at compile time
  - Easy to measure but not necessarily a good indication

- 2) The most used at runtine in the optimized code
  - Gives us the really useful optimizations

Demo, Conclusion & Questions

- Basic examples LLVM –O3 vs LLVM –O3 + Souper

- Things that Souper does but LLVM doesn't
- Sometimes Souper is of little use… (and why)

- Questions ?