Collaborative Compiler Vectorization

Jensen, Nicklas Bo; Probst, Christian W.; Karlsson, Sven

Publication date:
2014

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
Collaborative Compiler Vectorization
Nicklas Bo Jensen, Christian W. Probst and Sven Karlsson
Technical University of Denmark

Motivation
- Compilers are only partially successful at automatic vectorization
- Only 62 out of 151 inner loops automatically vectorized by GCC, 109 by at least one of the compilers
- Many vectorization obstacles can be handled by at least one of the compilers

![Diagram showing vectorization frequencies for GCC, PGCC, and ICC](image)

We argue that many compiler limitations can be overcome if the compilers collaborated!

Basic Idea
- Combine compiler feedback from many compilers
- Use feedback to make all compilers optimize further
- Apply automatic refactorings and validate effect
- Only advice on automatic refactoring if it has any effect
- Suggest automatic refactoring to the programmer and let the programmer determine whether the transformation is safe

![Diagram showing tool feedback loop](image)

Evaluation
- Combined 30 additional loops vectorized
- Speedups of up to 17x achieved on one loop using GCC

Future Work
- Take input from more compilers: XLC, Clang/llvm and Oracle Solaris Studio
- Include advice and optimization reports for automatic parallelization
- Provide feedback on more obstacles
- Use advice on other platforms, e.g. for automatic parallelization on GPUs

Conclusion
- We can make multiple compilers collaborate using optimization reports and automatic refactorings
- Cost effective total speedup of up to 5.8% on entire benchmark suite