# Interactive Analysis

Vincent St-Amour

April 12, 2013

## 1 Parascope

```
@article{parascope,
 author={Cooper, K.D. and Hall, M.W. and Hood, R.T. and Kennedy, K.
         and McKinley, K.S. and Mellor-Crummey, J.M. and Torczon, L.
         and Warren, S.K.},
 title={The ParaScope parallel programming environment},
 journal={Proceedings of the IEEE},
 year={1993},
 month={feb},
 volume={81},
 number={2},
 pages={244--263},
}


@article{parascope-experiences,
 author = {Hall, Mary W. and Harvey, Timothy J. and Kennedy, Ken
           and McIntosh, Nathaniel and McKinley, Kathryn S.
           and Oldham, Jeffrey D. and Paleczny, Michael H.
           and Roth, Gerald},
 title = {Experiences using the ParaScope Editor: an interactive
          parallel programming tool},
 journal = {SIGPLAN Not.},
 year = {1993},
 month = {jul},
 volume = {28},
 number = {7},
 pages = {33--43},
}
```

The first paper describes Parascope, a programming environment for writing parallel Fortran programs. It includes a parallelizing Fortran compiler, an editor and a debugger, all deeply integrated with each other. From the perspective of interactive analysis, the integration between the compiler and the editor is most interesting.

The Parascope editor displays some of the internal state of the compiler for programmers to see. Most importantly, it shows the results of dependence analysis (the most important analysis when it comes to parallelizing programs) and correlates these results with the program text. This gives programmers insight into the analysis process, and allows them to detect when analysis is being overly conservative.

When programmers notice unfeasible dependences resulting from analysis limitations, the Parascope editor allows them to edit the compiler's internal state and override the analysis. Dependences marked by the programmer as unfeasible are ignored by the compiler, which may allow it to perform optimizations that it previously ruled out. If the marked dependences are actually feasible—the programmer made a mistake—the compiler may end up performing incorrect optimizations.

Even when programmers determine that a given reported dependence is feasible, dependence information can still be useful. When programmers are notified about an unexpected dependence in their program, e.g. in a loop they were hoping to parallelize, they learn about potential missed optimization opportunities. They can then go and refactor their program to solve the problem and realize these optimization opportunities.

In the process of compiling programs, compilers generate a lot of dependence information. Each piece of information displayed by the editor comes at a cost in terms of programmer time; choosing which information to display is a key challenge when designing an interactive analysis system. Parascope addresses this challenge using two main techniques: progressive disclosure (information is only shown for the loops the programmer is actively working on) and user-defined view filtering.

The second paper reports on a 1991 workshop where the authors performed user studies using Parascope. They invited Fortran programmers from multiple industrial, academic and government sites to bring programs and to work on parallelizing them using Parascope during the workshop.

The results were positive: programmers successfully parallelized their programs using the development environment. However, the users reported that, since they did not fully understand why dependence analysis made its decisions, they did not always feel confident overriding its results.

## 2 Per Larsen's Thesis Work

```
@inproceedings{larsen-ieee,
 author={Larsen, P. and Ladelsky, R. and Lidman, J. and McKee, S.A.
         and Karlsson, S. and Zaks, A.},
 title={Parallelizing more Loops with Compiler Guided Refactoring},
 booktitle={International Conference on Parallel Processing},
 year={2012},
 month={sept.},
 pages={410--419},
}


@phdthesis{
 author={Larsen, P.},
 title={Feedback-Driven Annotation and Refactoring of Parallel
        Programs},
 school={Technical University of Denmark},
 year={2011},
}
```

The first paper, as well as chapters 7 and 8 of Per Larsen's dissertation, describe a tool for improving parallelization of C programs. Like Parascope, they accomplish this goal by including the programmer in the optimization process. The main goal of this work is to reduce the number of spurious reports shown to programmers. The authors accomplish this by only showing analysis failures that correspond to missed optimizations and by only showing those that are resolvable by programmers.

Using this tool, programmers first compile their programs. Then, the tool informs programmers of the resolvable missed optimizations it found and provides recommendations that may solve the problem. Programmers then edit their programs, taking the tool's output into consideration, and compile them again. This compilation cycle goes on until programmers are satisfied with the resulting optimizations.

The authors developed two techniques to determine whether failures are resolvable or not: *assistance* and *speculation*. The key insight behind assistance is that different parallelizing compilers, using different parallelization strategies, will succeed at parallelizing different sets of programs. Therefore, if one compiler fails at parallelizing a given program but another compiler succeeds, the failure is likely due to a limitation in the first compiler, and could be resolved by programmers.

The key insight behind speculation is that if a compiler fails at parallelizing a program, but succeeds when relaxing the correctness guarantees it provides (when passed flags that allow it to make possibly unsound assumptions, for example), then the failure is again likely due to a limitation in the compiler. Such failures are also likely to be resolvable by programmers.

The presented tool uses these two techniques to filter the information it provides to programmers. It compiles programs of interest using multiple combinations of differ-

ent parallelizing compilers and different compiler options, and analyses the resulting successes and failures to determine which failures are resolvable.

This approach has some advantages over the Parascope approach. First, since programmer feedback is encoded as part of the program (as opposed to being part of the compiler's internal state), it survives changes to the program. Second, the compiler has to approve the changes made by programmers and can reject optimizations that it judges unsafe. Finally, programmer time is used more efficiently because only actionable information is shown by the tool. On the other hand, this approach is less powerful than the Parascope approach: only changes that are expressible as source program changes are possible.