Compilers Laboratory

DCC
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

ICEx Instituto de Ciências Exatas

UFMG

Is it scientific?

Fernando Quintão

Which results can give us good papers?

1. We have re-implemented the entire gcc compiler in Java.
2. We show that tracking read-after-write dependences is enough to debug parallel code.
3. This paper presents the first open-source implementation of the icc compiler.
4. We present an alias analysis technique that is faster than previous approaches for languages without pointer arithmetics.
5. We provide experimental evidence that only 2% of all the loops written in C found in SourceForge are embarrassingly parallel.
6. We have implemented and tested a complete simulator for the x86 architecture.
7. We present a proof that the register allocation algorithm of Appel *et al* [Appel'94] is correct.
8. We show that the register allocation algorithm of Appel *et al* [Appel'94] is not correct in face of register aliasing.
9. We describe a new object-oriented programming language that combines features of Python and Java.

Not very publishable

1. We have re-implemented the entire gcc compiler in Java.
2.
3. This paper presents the first open-source implementation of the icc compiler.
4.
5.
6. We have implemented and tested a complete simulator for the x86 architecture.
7.
8.
9. We describe a new object-oriented programming language that combines features of Python and Java.

Simply implementing something is not a recipe for a paper. Can the world learn something with your implementation?

Unless you bring something new.

1. We have implemented and tested a complete simulator for the x86 architecture. This simulator is faster than previous approaches, and can run even on a smartphone having modest computing capabilities.
2. We have implemented and tested a complete simulator for the x86 architecture. This simulator makes it easy to design and test new instructions for this architecture. As a proof of concept, we have used it to reproduce clamp, and instruction proposed by Xiu *et al* [Xiu'13], and we show that the correct implementation of this instruction can be done with resources already in place in the x86, contrary to what was originally thought by Xiu *et al*.

Not very publishable

1. We present a proof that the register allocation algorithm of Appel *et al* [Appel'94] is correct.

Unless...

1. Appel et al. have left this proof as an open problem, whose solution has eluded the programming languages community thus far.
2. In the process of building this proof, we have uncovered a small inconsistency in the algorithm, which can be fixed by means of a small extension that preserves all the good properties of the original design.

Solid Contributions

1. We show that tracking read-after-write dependences is enough to debug parallel code.
2. We present an alias analysis technique that is faster than previous approaches for languages without pointer arithmetics.
3. We provide experimental evidence that only 2% of all the loops written in C found in SourceForge are embarrassingly parallel.
4. We show that the register allocation algorithm of Appel *et al* [Appel'94] is not correct in face of register aliasing.

A research paper must bring something **new**.

It can be an analytical observation:

- We show that tracking read-after-write dependences is enough to debug parallel code.

Or an empirical observation:

- We provide experimental evidence that only 2% of all the loops written in C found in SourceForge are embarrassingly parallel.

1. Refute old assumptions
2. Shows something surprising
3. Answer open-questions

You can beat previous approaches:

- We present an alias analysis technique that is faster than previous approaches for languages without pointer arithmetics.

Feels good to be on top of everything :-)

You can combine known-techniques in novel ways

- We combine range-analysis and points-to analysis to provide an alias analysis technique that handles pointer arithmetics in C.

You can introduce a problem

- We show a new way to attack software, which cannot be prevented by the current protection mechanisms.
  - E.g.: "ROP is Still Dangerous: Breaking Modern Defense"

You can debunk previous art

- We show that the register allocation algorithm of Appel *et al* [Appel'94] is not correct in face of register aliasing.
  - E.g.: "Register Allocation After Classical SSA Elimination is NP-Complete"
    - "Even though Hack et al. have shown that register allocation has polynomical time solution for SSA-form programs, we show that this result does not hold after the elimination of SSA-form, if this elimination is done using known-approaches."