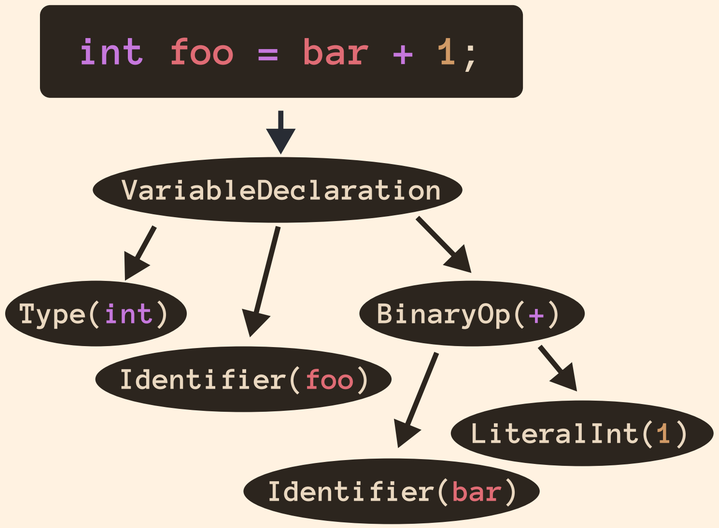
FINAL PRES.

Memory Layout Optimisation on Abstract Syntax Trees

Applying Data-Oriented Design to speed up the type-checking and code-generation compilation phases.

1. Introduction



ASTs are created during parsing phase, and utilised during type-checking & code-generation. Naive AST implementations are **sparsely laid out in memory**, resulting in **bad cache locality**. Zig compiler devs **re-implemented** their AST, leading to 40% performance improvements [1].
This research leverages these findings.

2. Research question

How does the application of **Data-Oriented Design** principles [2] on **Abstract Syntax Trees** affect the **speed** of the **type checking** and **code generation** phases for compilation of procedural programming languages?

3. Methodology

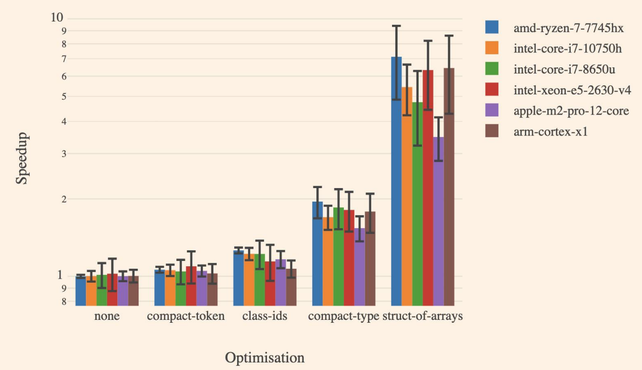
Benchmarking approach:

1. Collected dataset of preprocessed **C code**.
2. **Transpiled** to Tea language.
3. Implemented various **AST layout optimisations** for Tea compiler using DOD principles.
4. **Benchmark** performance for type-checking and code-generation phases.
5. Data **analysis** => conclusion & recommendations.

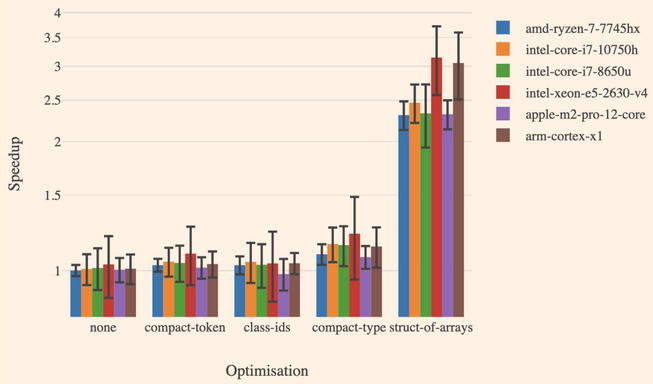
4. Results

- **Type-checking:**
  - SoA 3.5x - 7.1x speedup.
  - Most gains in this phase.
- **Code-generation:**
  - SoA 2.3x - 3.1x speedup.
- Significant **upwards trend**.
- **Positive** impact **cache** miss rate & **memory** usage.
- AST **size** 6-12x smaller.

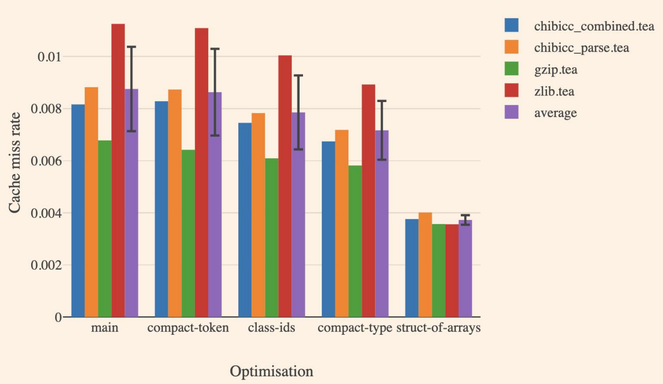
Average speedup of AST optimisations (type-check phase) by machine



Average speedup of AST optimisations (code-gen phase) by machine



Cache miss rate (LLd)



5. Conclusions

- **Struct-of-Arrays** signif. speedup in both phases.
- SoA impl. required **extensive development**.
- **Simple** DOD optimisations => **modest** speedups.
- **Variations** in results by **programs** & **machines**.
- **No** significant performance **reductions**.

Recommendations:

- **Adopt SoA** for substantial speedups, if **development costs** can be justified.
- **Compact** data structures, apply **DOD** principles.
- Use memory pools & efficient **allocation**.

6. Future work

- **Advanced** memory layout & allocation strategies.
- **Dynamic** AST optimisations (**JIT**)
- Rework AST impl. in **production-grade** compilers.
- **Other** types of programming **languages**.

References

[1] *0.8.0 Release Notes - The ZIG Programming Language.*

[2] *Data-oriented design*. Stockport, England: Richard Fabian, Sept. 2018.

Author: Iannis de Zwart <i.r.e.dezwart@student.tudelft.nl>

Supervisors: Soham Chakraborty, Dennis Sprokholt

TU Delft