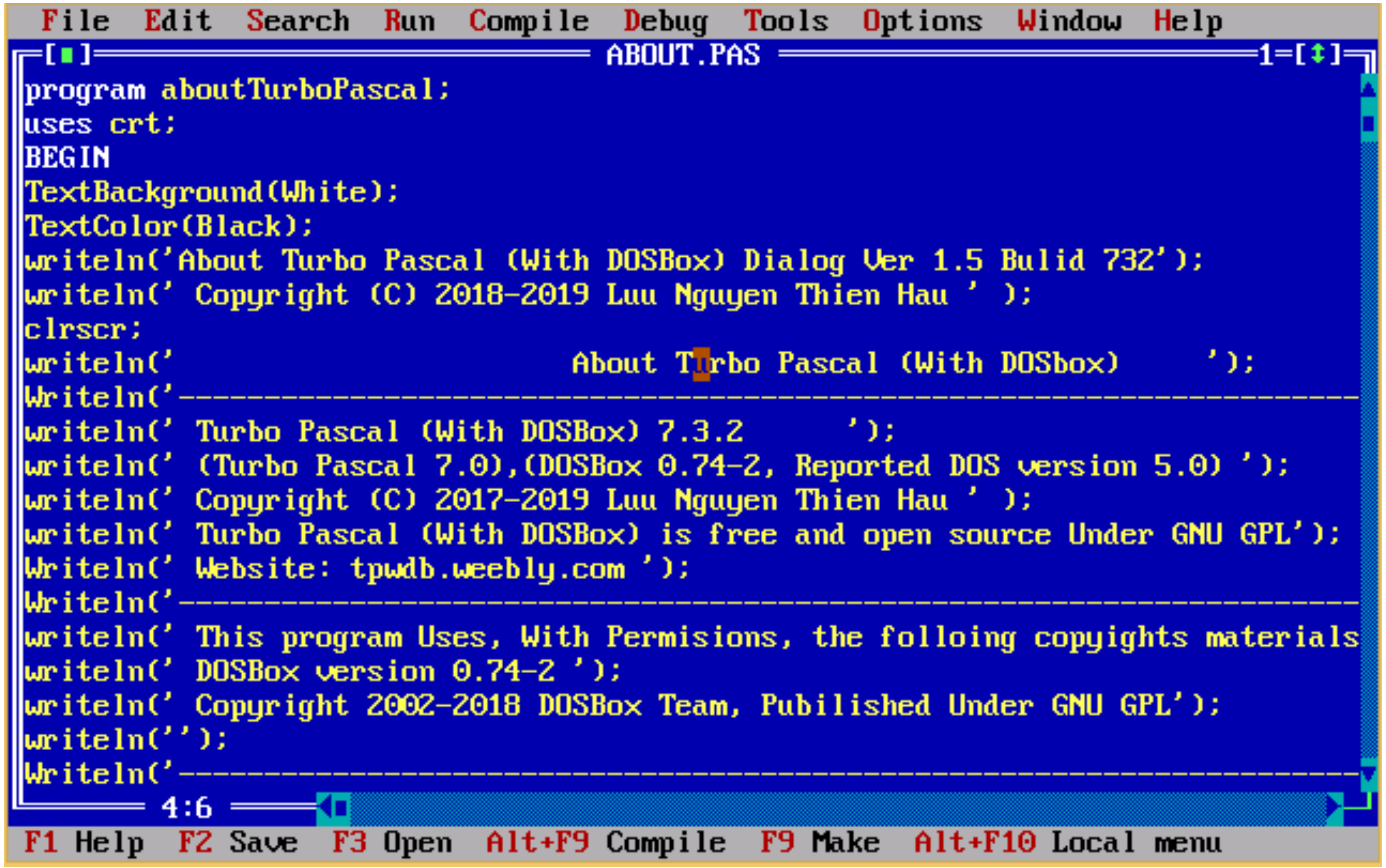
PROGRAM ANALYSIS:
A VETERAN COMPILER WRITER'S PERSPECTIVE

Sun Chan

TURBO PASCAL



$49.95 (Year 1983)

The first commercial compiler for the masses

1st COMMERCIAL COMPILER WITH GLOBAL OPTIMIZATIONS

Engineering of a RISC Compiler System, 1986 (MIPS)

- Full optimizations through out the entire compilation
  - Link time optimization (inter-procedural)
    - Literal merging
    - GP relative
    - Variable promotion (global → static → local)
  - Instruction scheduling (local)
  - Partial Redundancy Elimination (global)
  - Strength Reduction (global)

Global optimization data flow equation

```
OUT[ENTRY] = ∅;
for (each basic block B other than ENTRY) OUT[B] = ∅;
while (changes to any OUT occur)
    for (each basic block B other than ENTRY) {
        IN[B] = ∪_{P a predecessor of B} OUT[P];
        OUT[B] = gen_B ∪ (IN[B] − kill_B);
    }
```

M * N^2 algorithm

Figure 9.14: Iterative algorithm to compute reaching definitions

1. Compute live-in/live-out *locally* (within basic block) - *N^2*
2. Compute live-in/live-out *globally* based on result from step 1

Global optimization data flow equation

```
OUT[ENTRY] = ∅;
for (each basic block B other than ENTRY) OUT[B] = ∅;
while (changes to any OUT occur)
        for (each basic block B other than ENTRY) {
              IN[B] = ∪_{P a predecessor of B} OUT[P];
              OUT[B] = gen_B ∪ (IN[B] − kill_B);
        }
```

M * N^2 algorithm

Figure 9.14: Iterative algorithm to compute reaching definitions

Until 1995 ALL global optimizers used bit-vector implementation

- Each bit represents one variable (or expression)

- Bit-vector implementation is mostly flow insensitive
- Global optimizer is SLOW and hard to debug
- Debug optimized code is hard
  - insertion and deletion of code (optimization process) is independent

PARTIAL REDUNCANCY ELIMINATION –
THE MOTHER OF ALL SCALAR OPTIMIZATIONS

- Bi-Directional data flow equation problem, no loop analysis needed
  - **19** data-flow equations to solve
  - Flow sensitive, N^3 complexity
- Subsumes most of the important optimizations in one single phase
  - Dead code elimination
  - Strength reduction (as combined algorithm)
  - Code hoisting
  - Partially redundant code
  - Loop invariant code motion
  - Common subexpression elimination

ENTER SSA ERA IN 1988

Global value numbers and redundant computations, 1988 - Barry Rosen, …

- Dependency (use-def analysis) from “dense” to “factored”
  - Iterative and sparse algorithm possible – On demand
  - Flow sensitive is implied with SSA
  - Most known optimizations are now linear (or close to)
  - Can be used in code generator optimizations
    - Live range analysis
    - register coloring

- Partial Redundancy Elimination using SSA still elusive
  - Original PRE algorithm needs to solve 19 data flow equations
  - Only on scaler variables
    - Arrays and class objects are not covered

HSSA, 1996 – (MipsPro / Open64)

- Arrays, class objects and alias incorporated into SSA

TAMING THE TIGER – SSAPRE

Lazy code motion, 1992 – Knoop et al

- Reduce PRE from bi-directional to uni-directional
- Provides the inspiration to the solution

A new algorithm for partial redundancy elimination based on SSA form, 1997 – (MipsPro / Open64)

- Bit vector is eliminated in global optimizers
- Other extensions to speculative code motion, parallel optimization, …

- Other more important benefits
  - Optimizer is FAST (almost linear)
  - Flow sensitive global optimization
  - Debug optimized code is SOLVED
  - Debug optimizer problems can be AUTOMATED

☞ Almost ALL compilers now uses SSA and some variation of SSAPRE

WHAT ABOUT DYNAMIC LANGUAGES?

- Can Java optimization use SSA?
- Can all dynamic behavior be resolved at static time?
- Even at linear complexity, when “N” is large, is that acceptable?
  - JIT
  - AOT
- At static compile time, can we reduce apparent dynamic behavior to static?
  - Type inference
  - De-virtualization

☞ *It is all about increasing scope of analysis*

INTER-PROCEDURAL OPTIMIZATION HISTORY

Cross Module Optimizations , Its implementation and benefits, 1987 – MIPS

- Link time optimization (literals merging, …)

FIAT: A framework for Interprocedural analysis and transformation, 1995 – Mary Hall

Optimizing the performance of dynamically-linker programs, 1997 – (MipsPro / Open64)

- Link time optimization with shared objects
- IPA with summary info, call graph, interprocedural alias analysis

INTER-PROCEDURAL ANALYSIS (IPA) FEATURES

- *Local* analysis generate "Local Summary" for each procedure and call sites
  - Immediate effects for particular data flow problems
- *Global* analysis phase queries "Local Summary" for each data flow problem to be solved
- Same complexity problem like that of bit-vector global optimizers
  - With "M" much larger due to larger scope, memory and CPU time grow exponentially

IPA - THE DEVIL IS IN THE DETAILS

- Summary info is a constantly changing set of information depends on each application/benchmark case
  - Variety and volume becomes maintenance nightmare
- SSA is local within a function
  - Flow sensitivity is lost at call boundaries
- Parallel applications are not properly represented in a “call-graph”
  - Threaded programs (e.g. Java)
- IPA analysis implementations have a habit of running out of memory
  - Problem is too large for every commercial implementation that I know of
  - Apps are getting larger as IPA compilation also wants to be more aggressive

HISTORY OF *IPA* IMPLEMENTATION

1987

1997

2007

2017

1[st] generation: Link time optimization (LTO)

1987 - MIPS

2[nd] generation: IPA + LTO

1997 - MipsPro /Open64

2000 - Intel

2002 - HP

2006 - GCC (single file, LTO)

3[rd] generation

?

INTER-PROCEDURAL ANALYSIS – THE 3rd GEN

Deep analysis for Java, Python, …, security defects

爱科识
xcalscan

- Inline is not the primary weapon to get performance or analysis precision
- Cross procedural SSA
    - Side effects captured in SSA U-D chain
- Sparse on demand analysis
- IPA on Java program (increase analysis scope with no aggressive inline)
    - Static analysis or AOT for “static” behavior e.g.
        - If ”life-time” truly ends, ”insert delete”
        - Code motion on “locks” that are not optimal
    - Annotate “likely points” to help runtime speedup or precision such as
        - Class loading that can be delayed
        - GC hints

QUESTIONS?