CMPSCI 630 Systems **Spring 2023**

Lecture 3:Compiler Optimizations

Date: Feb 14

Lecturer: Emery Berger *Scribe: Walid A. Hanafy*

3.1 Register Allocation

Register allocation is a key problem in designing compilers. Registers are super fast, but are available in limited numbers.

Example:

Operations on Registers (A, B)
ADD EAX, EBX, ECX, $C \leftarrow A + B$

In the following example, we can see that the limitations of register might affect the performance.

```
void do_something(){
    int a,b,c;          // Can fit in registers
    float x,y,z;        // Can fit in registers
    float arr[1024];    // Cannot fit in registers (may be even cache memory)
}
```

Registers usally are accessed with 1 Cycle. In comparison, the L0 Cache require 5–7 Cycles.

3.1.1 Compiler Role in Register Allocation

Compilers play an important role in register allocation optimizations by trying to:

1. Identify Live Variables. i.e. Identifying the start and the end of each variable usage.
2. Maximize number of variables in registers over time.
3. Minimize "Spills". Spilling is moving a variable from a register to memory.

The Register Allocation problem is reducible to graph coloring, so it is NP-Complete. However, even if there exist a polynomial time algorithm, the process is still too slow especially for JIT compilers.

3.1.2 Just In Time (JIT) Compilation

JIT is a background compilation, on demand, for "hot" code (code that executes often). JIT uses a compiler and an interpreter: code is interpreted until the virtual machine decides it is worthwhile to compile.

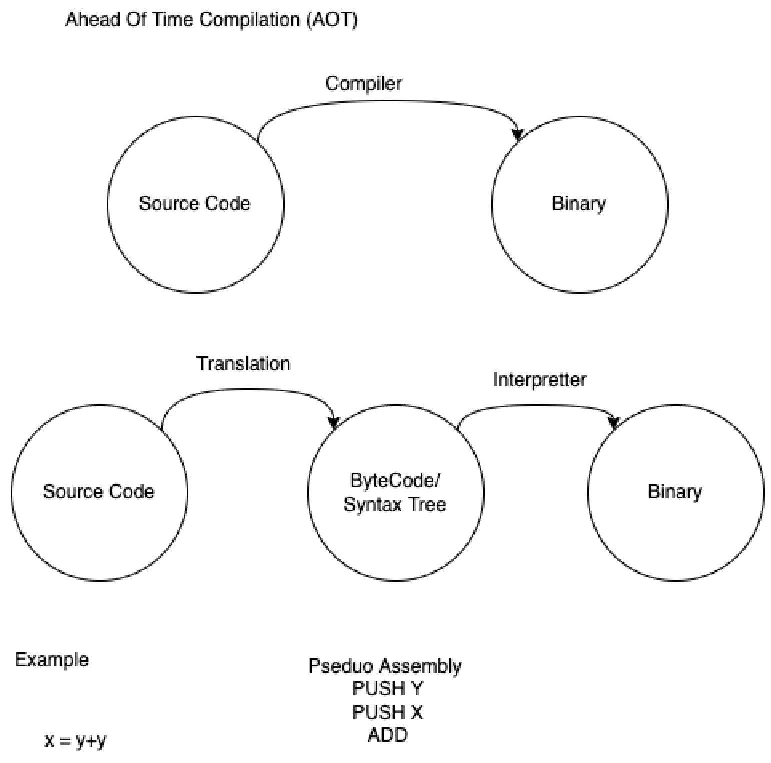

Figure 3.1: Type of Compilations

Checkout: `https://accidentallyquadratic.tumblr.com/`

Solution The register allocation process requires expensive analysis. They are very difficult to use in large code bases and for JITs. The solutions usually use greedy heuristics, which often try to reduce *register pressure.*

Note:

Web Assembly: The authors of webassembly tried to solve thread management by dedicating a register for thread location data and garbage colection by dedicating a register to the root (Pointer to root). The solution created a register pressure that made webassembly 30% slower than native.

The following figure explains the mark-sweep GC technique as well as how webassembly utilized the register.

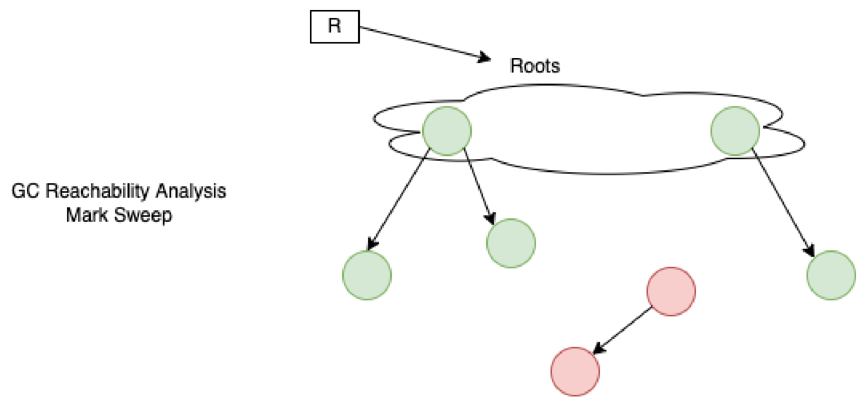

Figure 3.2: Reachability Analysis

3.2 FORTRAN is Alive!

Note:

APL ("Array Programming Language") introduced the notation for list comprehension:

```
arr = [x * x for x in range(100)]
```

Created by Kenneth Iverson at IBM.

FORTRAN is very influential as it allows for automatic parallelization, including loops. FORTRAN is used in the `numpy` Python package, which depends on BLAS "Basic Linear Algebra Subprograms" and LPACK "Linear Algebra PACKage".

Note:

The automatic ability to parallize code in FORTRAN made code written in it faster than that written in C. For example, after translating code from FORTRAN to C using F2C library it became 20-40% slower.

Languages like C/C++ are only able to parallelize code using special constructs and libraries like pthreads, MPI, OpenMP, etc.

3.2.1 Why FORTRAN Allows for automatic parallelization?

Same argument allows for SQL.

Note:

What is turing complete? A language that can be used to program any problem, at least in theory.
Non-Turing Complete Languages: SQL, Excel, Datalog, regex, ..
Turing Complete Languages: All general purpose programming languages
Most Domain Specific languages are not turing Complete.

Turing Completeness can be achived by simple constructs.

Example:

Read Tape, Write 0, Write 1.
If R0 go left, if not R0 go right, If R1 go left, If not R1 go right.

Note:

Low-level languages are hard to optimize as the intent is not clear.
High-level language are easier to compile as the intent is clear.
Peephole optimization ”Replacing instructions with more efficient instructions”, which is the last step in compiler optimization, applies to both.

So, back to our question why FORTRAN code can be parallelized? Because it doesn't have pointers, which means no variables could point to the same memory location. Therefore, no "race" conditions happen and the parallel must be equal to the sequential. Example codes include:

```
void matrixvec(float *m, float *v, float *r, int x, int y){
    // A code that does matrix vec multiplication
}
```

Although this code is "Embarrassingly parallel", speed up is linear with number of added nodes, since loops results are independent. The compiler can't fully guess whether there is some dependency or not. The compiler can be informed to treat them as independent by using the 'restrict' keyword.

Compiler usually try to find such cases by disambiguating pointers, pointer analysis, alias analysis, nested loop optimizations, inter-procedural analysis, flow sensitivity, unification, path sensitivity, branch prediction, etc.

The situation in which multiple pointers point to the same data is known as *aliasing*.
See: `https://en.wikipedia.org/wiki/Aliasing_(computing)`

Quote:

"Good Ideas might have ramifications" – A theme common in our course.

Note:

Path sensitivity is not scalable and cases 2^n explosion. Ex:

```
if ()
    if()
        if()
            if()
```

FORTRAN is a High level language for scientists.

1. Portable

2. Readable
3. Fast

3.3 LISP

Symbolic data structures:

1. Supports lists
2. Nested
3. Heterogenous

```
("Hello" 5 (name "emery" "height" 64)) // looks like json
```

LISP supports linked lists out of the box. FORTRAN doesn't support pointers and hence linked list is not supported. The language can be used to implement linkeds list by using offset list as shown below.

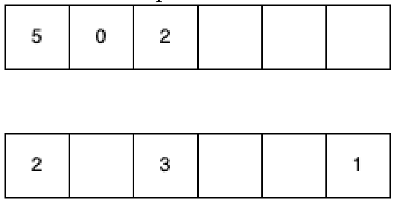

Figure 3.3: Offset List

Note:

LISP was used to build “SHRDLU” an NLP and 3D simulator tool.

3.4 Garbage Collection

Garbage collection is the automation of memory management. The two main techniques are 1) Reference Counting and 2) Mark-Sweep.

3.4.1 Reference Counting

This is a method is a continuos way of Garbage Collection.

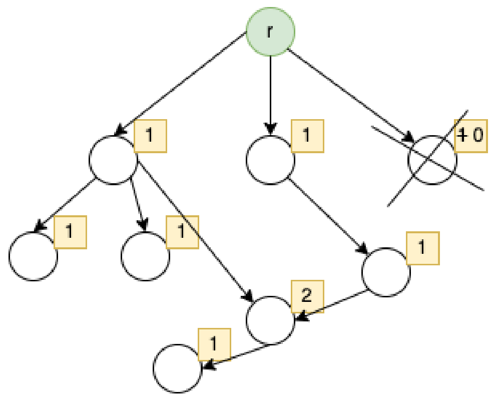

Figure 3.4: Offset List

RC can leak memory in the case of cycles. In the next example, when X is removed the branch it was pointing to should be removed.

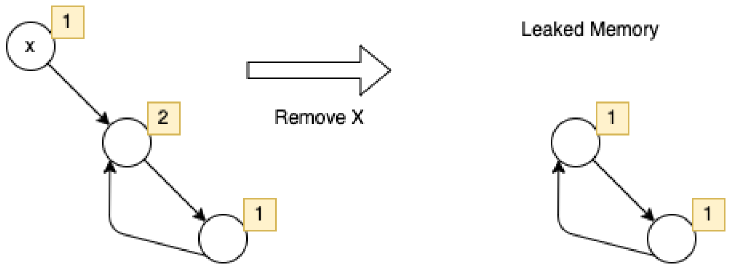

Figure 3.5: Reference Counting Can leak memory

3.4.2 Mark-Sweep

The mark-sweep GC works by iterating over all allocated memory objects and marking as reachable or not, as shown in figure 3.2.

Python uses Reference Counting and Mark-Sweep.

GC languages

Java, C#, GO, Lisp, ruby, DART, Python, Kotlin, TCL, Swift, OCamel, Perl, Modula, ADA.

Non-GC languages

RUST, C/C++, FORTRAN, ALGOL, PASCAL.