Improving the Efficiency of CHA through Parallelization

Miguel Velez and Jason Sawin

University of St. Thomas. St. Paul, MN

College of Arts and Sciences
Computer & Information Sciences

Interprocedural Analysis

Interprocedural analysis is a software engineering technique that uses calling relationships among procedures to analyze an entire program. Researchers and software engineers rely on such analyses to help them build efficient and safe programs. There are several algorithms used in this type of analysis that statically analyze source code to determine the calling relationships and method reachability in a program. The following algorithms are some of the most studied and important processes in the field: naïve, CHA[1], RTA[2], and points-to analysis[3].

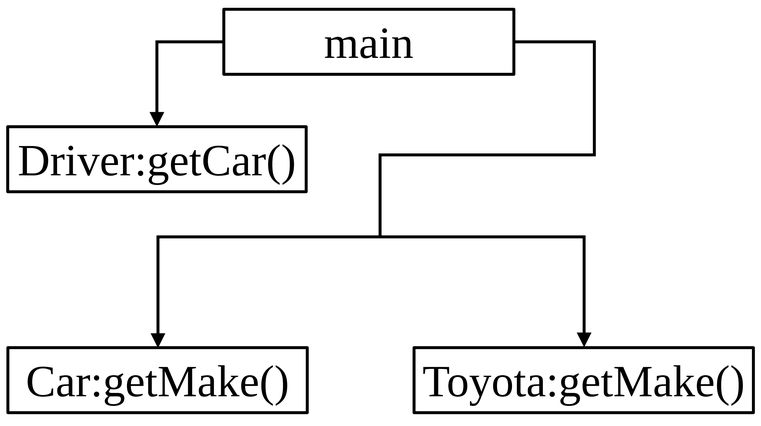
CHA

Fastest and most unreliable algorithm. Uses the type of a variable, together with the class hierarchy.

```
public class Car {
    private Stirng make;
    ...
    public String getMake() { return this.make; }
    ...
}
```

```
public class Toyota extends Car {
    ...
    public String getMake() { return "Toyota"; }
    ...
}
```

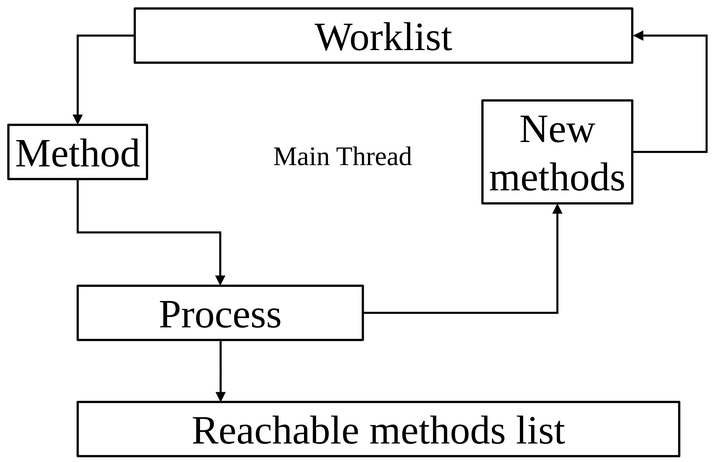
```
public class Driver {
    public static void main(...) {
        Car c = getCar();
        c.getMake();
    }
    public static Car getCar() { ... }
}
```



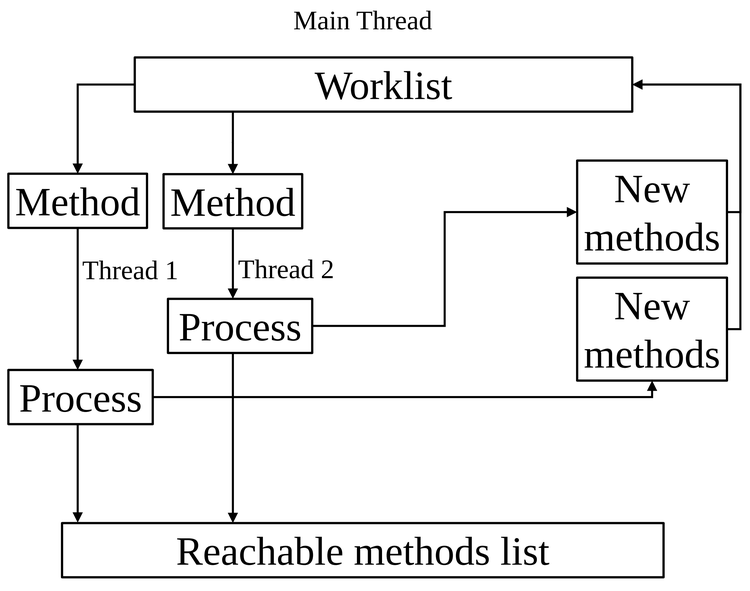
The call graph allows us to know which methods are reachable from a specific method. The type of variable *c* is not known. Therefore, both *getMake()* methods are included. For larger programs where a graph could have many edges, a list of reachable methods can be returned.

Parallel Algorithm

Reachability algorithm:



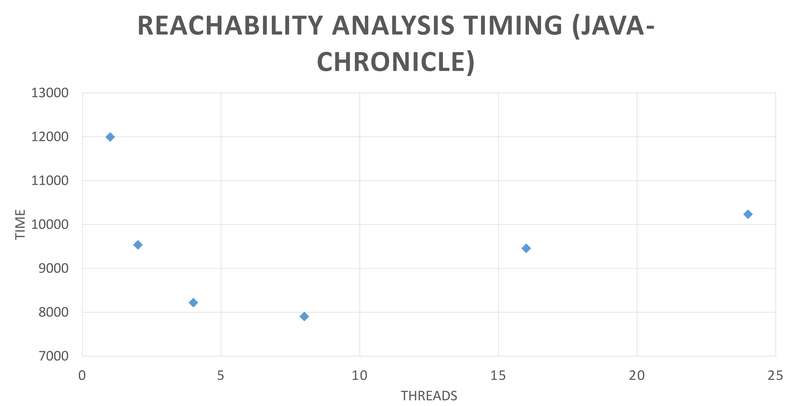
Parallel algorithm:

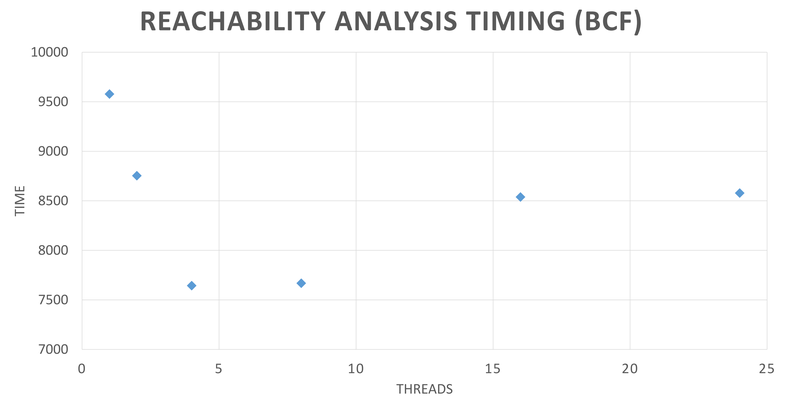


- Main thread creates threads that wait for methods to be analyzed and checks the worklist for new methods to dispatch.
- When a method is available in the worklist, a child thread analyses it.
- When the worklist is empty, the main thread signals each child thread to terminate. All methods in the program were analyzed.
- The worklist, reachable methods list, and several other parts of the algorithm are shared resources. They were synchronized to avoid concurrency errors.

Results

Tested algorithm in various Java programs





- Took advantage of modern multi-core architectures.
- As expected, the more threads used in the analysis, the less time it takes to complete it.
- There is an expected threshold. After a certain number of threads, the management of them done by the OS slows down the analysis.

Bibliography

[1] J. Dean, D. Grove, and C. Chambers, “Optimizations of object-oriented programs using static class hierarchy analysis,” in European Conference on Object-Oriented Programming, 1995, pp. 77–101.

[2] D. Bacon and P. Sweeney, “Fast static analysis of C++ virtual function calls,” in Conference on Object-Oriented Programming Systems, Languages, and Applications, 1996, pp. 324–341.

[3] A. Rountev, B. G. Ryder, and W. Landi, “Data-flow analysis of program fragments,” in ACM SIGSOFT Symposium on Foundations of Software Engineering, ser. LNCS 1687, 1999, pp. 235–252.