Program Level Performance Analysis

- Execution time of a program often varies with the input data values because those values select different execution paths in the program.
- Cache has a major effect on program performance.
- Execution times may vary even at the instruction level.
- Floating-point operations are the most sensitive to data values, but the normal integer execution pipeline can also introduce data-dependent variations.

Program performance is measured in following ways :

1. Simulator of CPU supplied by manufacture.
2. Timer connected to the microprocessor bus can be used to measure performance of executing sections of code.
3. A logic analyzer can be connected to the microprocessor bus to measure the start and stop times of a code segment.

Elements of Program Performance

Program execution time is given as

Execution time = Program path + Instruction timing

The path is the sequence of instructions executed by the program. The instruction timing is determined based on the sequence of instructions traced by the program path, which takes into account data dependencies, pipeline behavior and caching.

Program execution times depend on several factors :

1. **Input data values** : Different values, different execution paths.
2. **Cache behavior** : Also dependent on input values.
3. **Instruction level** : Floating-point operations and pipelining effects.

Program paths offer insight into a program's dynamic behavior that is difficult to achieve any other way. Unlike simpler measures such as program profiles, which aggregate information to reduce the cost of collecting or storing data, paths capture some of the usually invisible dynamic sequencing of statements.

Examination of programs' paths has unveiled a striking degree of path locality, which the computer architecture and compiler communities have profitably exploited to increase program performance.

Paths are useful at another level : as a convenient abstraction for reasoning about a program's runtime behavior.

A program path records a program's executable statements in the order in which they rim. For example, consider a simple function to add the even natural numbers from 1 to N :

```
int AddEvenNumbers (int N)
{
int sum = 0;
/* SI * /
for (int j = 1, j < = N ; j + = 1) {
/* S2 * / if ((j % 2) = = 0) {
/* S3 * /
sum += j
}
}
/* S4 * /
return sum ;
}
```

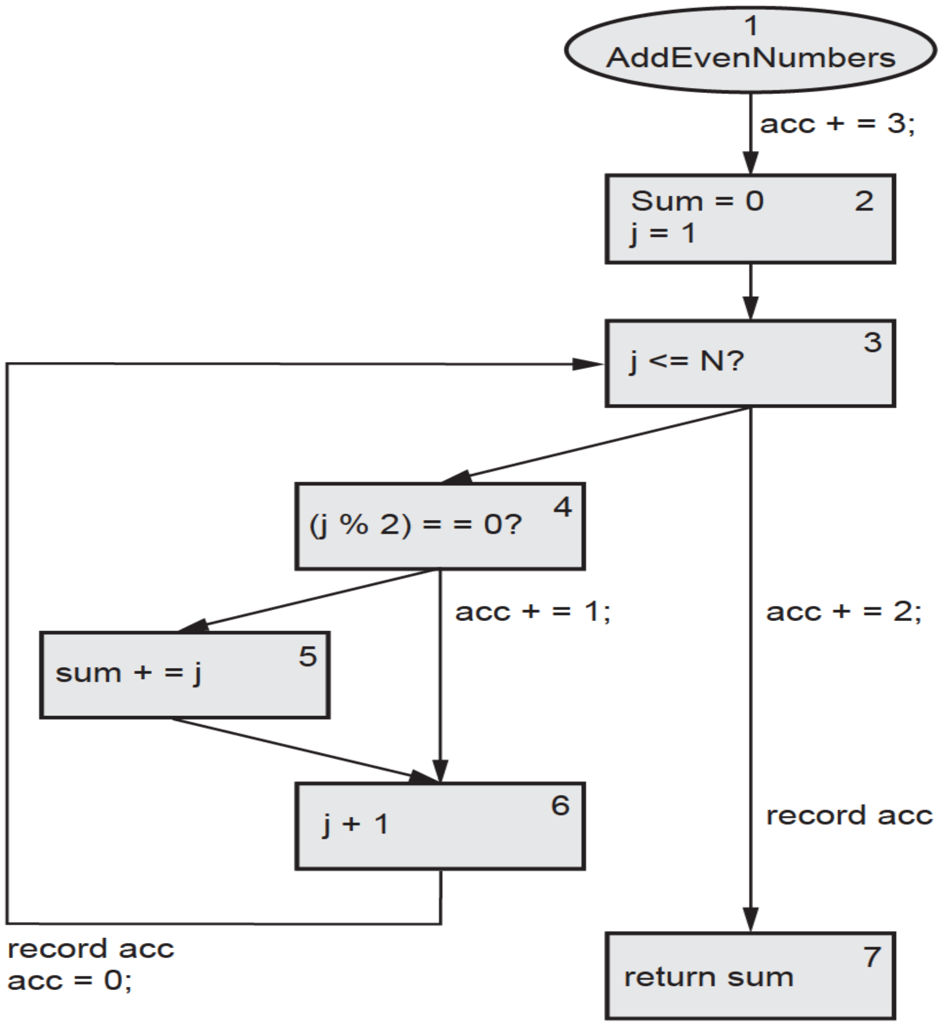
When invoked with an argument of 3, this function executes the path SI, S2, SI, S2, S3, SI, S2, SI, S4.

This type of path, which is also known as an instruction or statement trace, is unwieldy and difficult to manipulate for two reasons: first, its length is proportional to how long a program runs and, second, it must be read sequentially, like a magnetic tape.

Computer architects use instruction traces to simulate processor designs, but most others found that the cost of collecting and recording a full instruction trace outweighed its utility.

Path profiling code for the Add Even Numbers function. The code along the edges computes unique numbers for each acyclic path in the function.

Path	Value in acc
3, 4, 5, 6	0
3,4,6	1
3,7	2
1, 2, 3, 4, 5, 6	3
1, 2, 3, 4, 6	4
1, 2, 3, 7	5



Path profiling code

Program profiling, a widely used substitute for paths, captures which statements execute, but not the order in which they rim. Since profile only records statement executions, it can be compactly summarized as a table of execution frequencies. The preceding example's program profile is SI = 4, S2 = 3, S3 = 1, S4 = 1.

Program performance metrics :

1. **Average-case execution time** : Typically used in application programming.
2. **Worst-case execution time** : A component in deadline satisfaction.
3. **Best-case execution time** : Task-level interactions can cause best-case program behavior to result in worst-case system behavior.