

An LLVM based Loop Profiler

Shalini Jain^{*}, Kamlesh Kumar⁺,
Suresh Purini^{\$}, Dibyendu Das[£],
Ramakrishna Upadrasta^{*}

Indian Institute of Technology, Hyderabad^{*}
National Institute of Technology, Manipur⁺
International Institute of Information Technology, Hyderabad^{\$}
AMD India Pvt. Ltd[£]

Profiling

Profiling: A way to calculate run-time information

- Execution-time, Cache-misses, Iteration Count, etc ...
- Helps to analyze the code to fix performance related issues
- Need to do instrumentation to calculate profile information

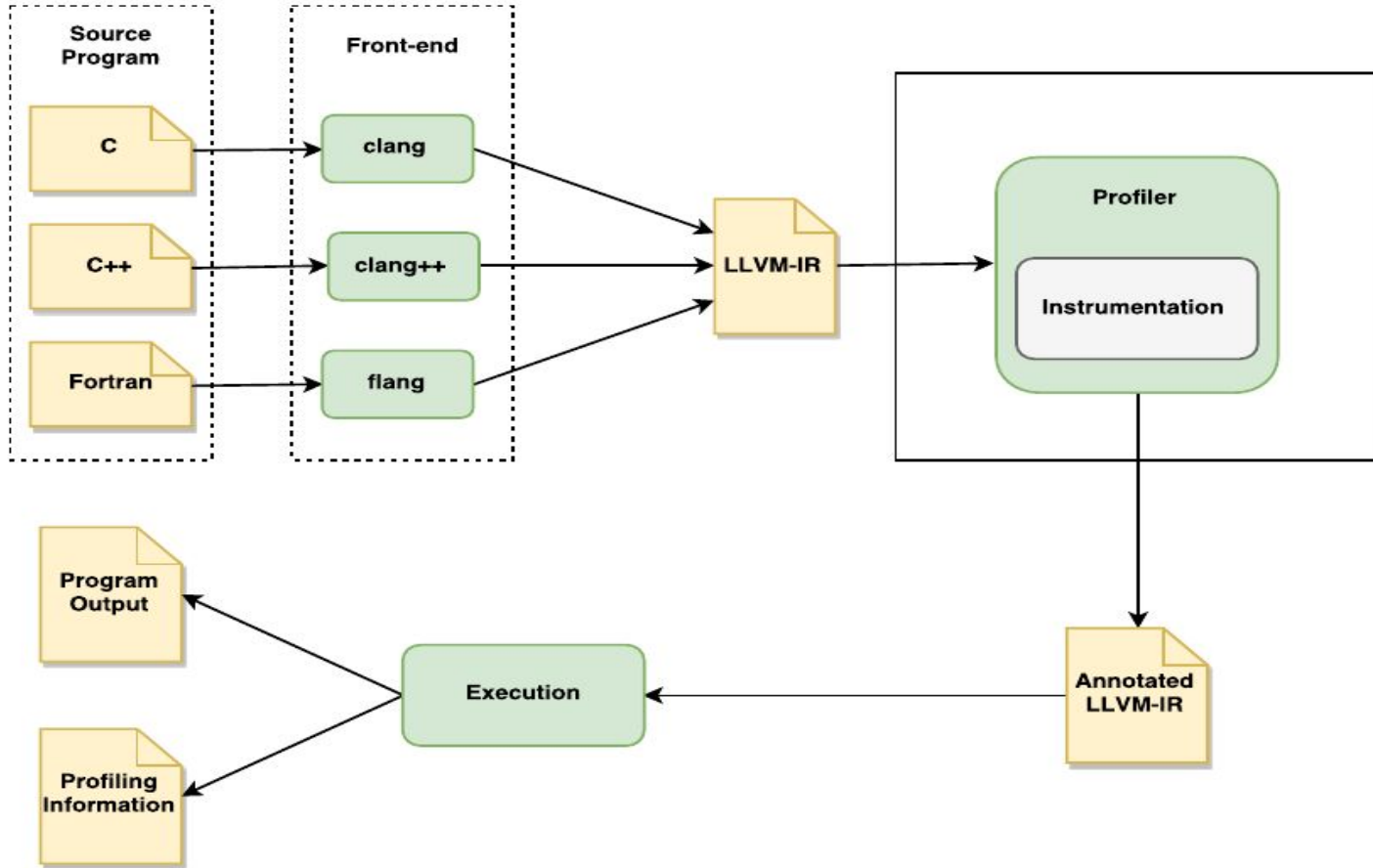
Currently: No Loop Profiler in LLVM

- For analyzing run time metrics

Our Contribution: Implemented an Loop based Profiler

- Calculates clock ticks
- Calculates iteration count

An LLVM based Loop Profiler: Flow Graph



Implementation

- Instrumentation For Each Loop
 - At end of pre-header block
 - Appended Instructions for first Call Instruction to clock function

```
%0:  
br label %1
```



```
%0:  
%1 = call i64 @clock()  
store i64 %1, i64* @.t1  
br label %2
```

Loop Pre-Header

Implementation

- Instrumentation For Each Loop
 - Before First Non ϕ Node of All Possible Exit Blocks
 - Append instructions for
 - Second Call Instruction to clock function
 - Store Difference of Two calls
 - Add current difference with previous value and Store it

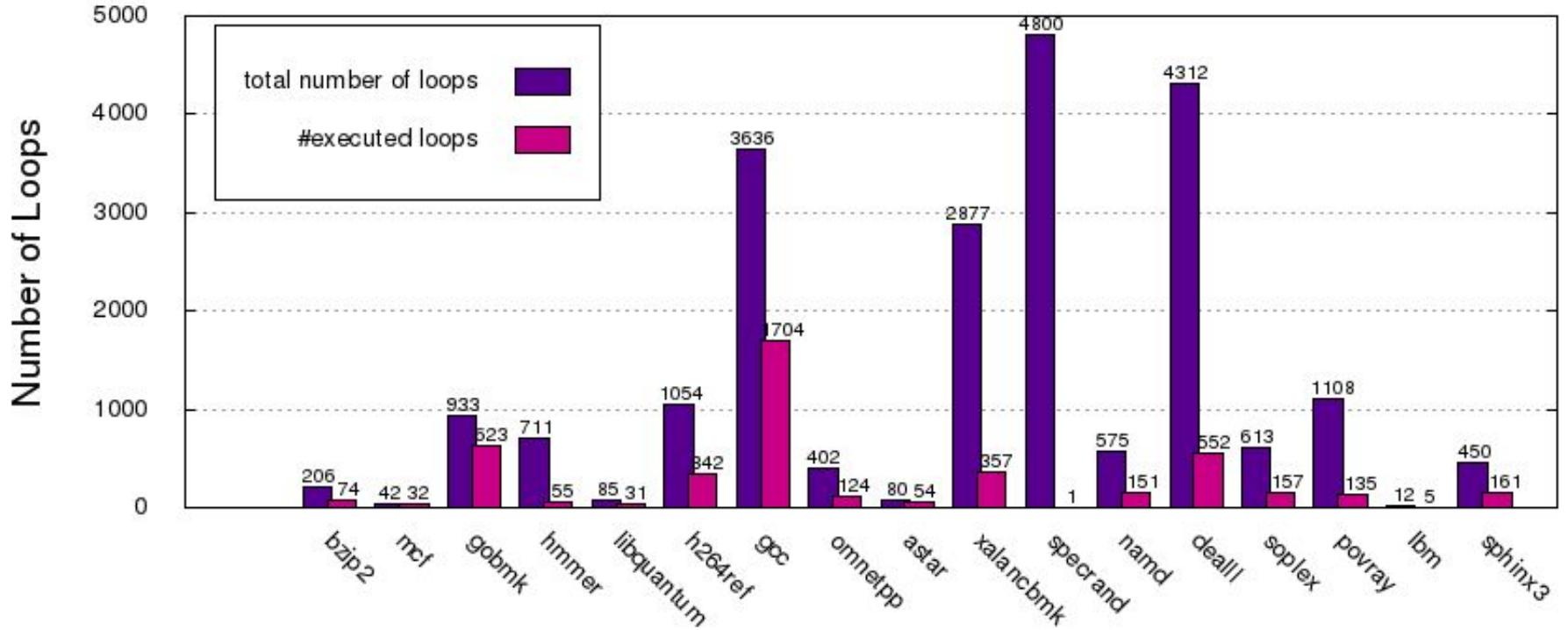
```
%7:
ret i32 0
```



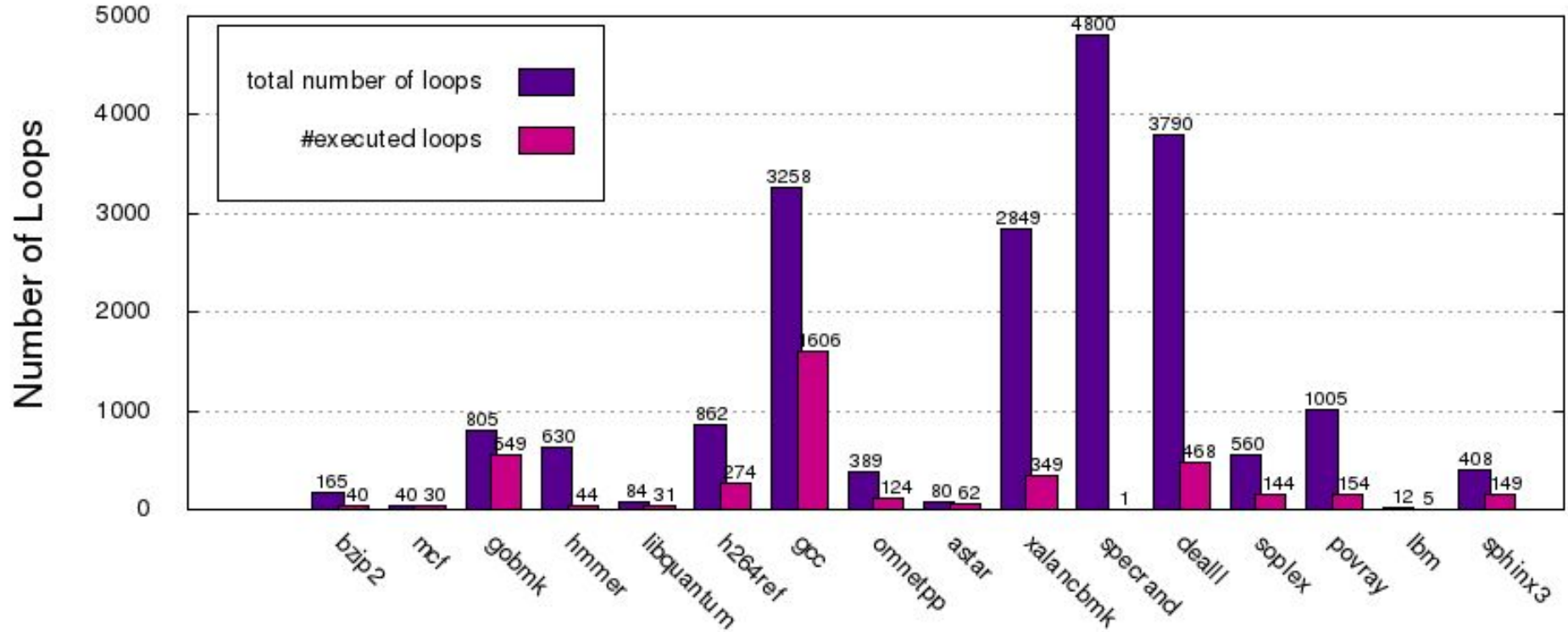
```
%8:
```

```
%9 = load i64, i64* getelementptr inbounds ([1 x i64], [1 x i64]* @cd, i64
... 1, i32 0)
%10 = add i64 %9, 1
store i64 %10, i64* getelementptr inbounds ([1 x i64], [1 x i64]* @cd, i64
... 1, i32 0)
%11 = call i64 @clock()
store i64 %11, i64* @.t2
%12 = load i64, i64* @.t2
%13 = load i64, i64* @.t1
%14 = sub i64 %12, %13
%15 = load i64, i64* getelementptr inbounds ([1 x i64], [1 x i64]* @t, i64
... 1, i32 0)
%16 = add i64 %15, %14
store i64 %16, i64* getelementptr inbounds ([1 x i64], [1 x i64]* @t, i64 1,
... i32 0)
%17 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x
... i8]* @.counter, i32 0, i32 0), i32 0)
%18 = load i64, i64* getelementptr inbounds ([1 x i64], [1 x i64]* @cd, i32
... 0, i32 0)
%19 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x
... i8]* @.counter, i32 0, i32 0), i64 %18)
%20 = load i64, i64* getelementptr inbounds ([1 x i64], [1 x i64]* @t, i32
... 0, i32 0)
%21 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x
... i8]* @.str1, i32 0, i32 0), i64 %20)
ret i32 0
```

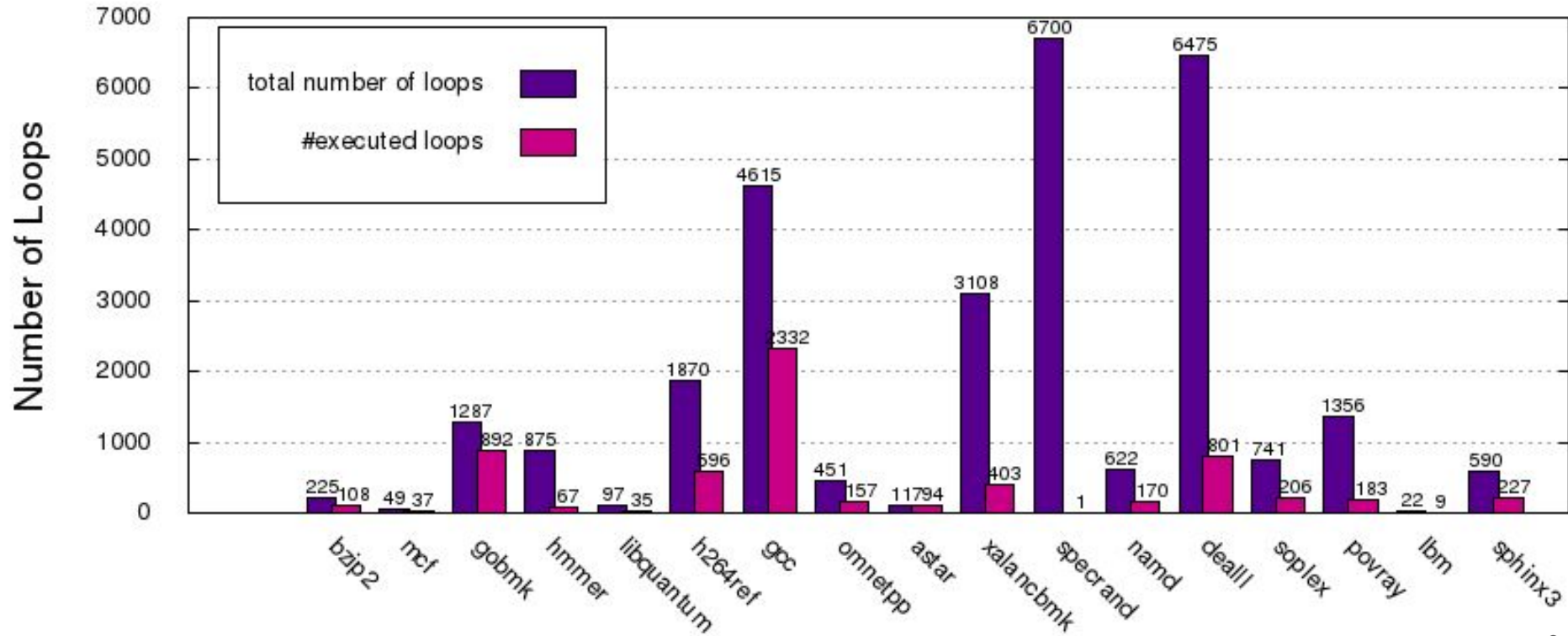
Results: SPEC CPU 2006 (Inner Loop)



Results: SPEC CPU 2006 (Outer Loop)

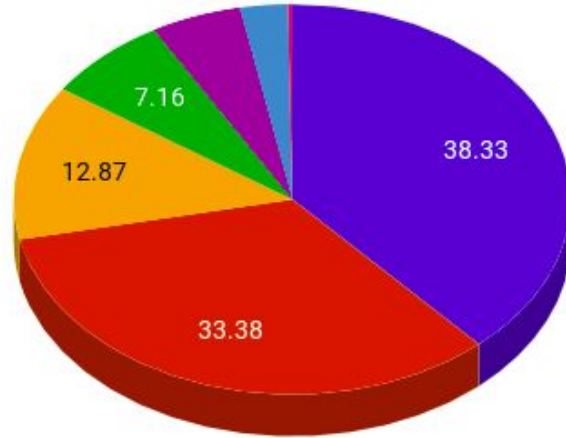


Results: SPEC CPU 2006 (All Loops)

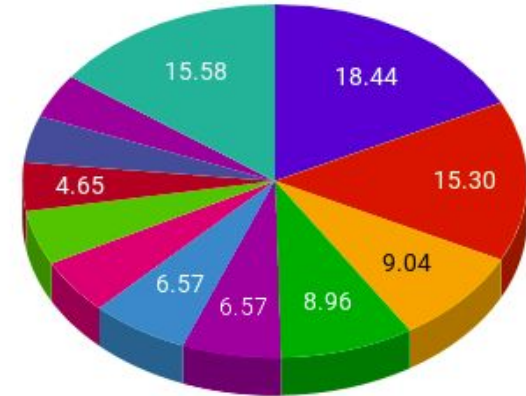


Result Analysis: SPEC CPU 2006 (INT)

hmmmer (SPEC 2006 INT)



xalancbmk (SPEC 2006 INT)

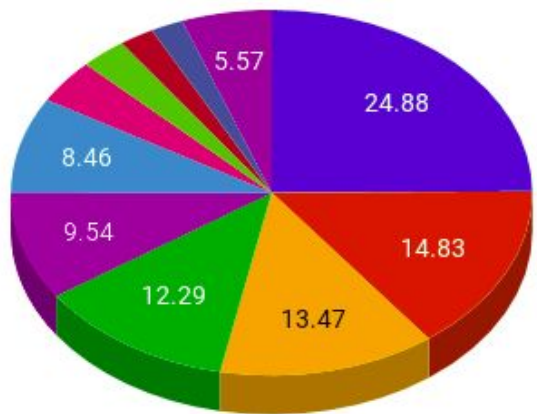


● Loop-1
 ● Loop-2
 ● Loop-3
 ● Loop-4
 ● Loop-5
 ● Loop-6
 ● Loop-(7-711)

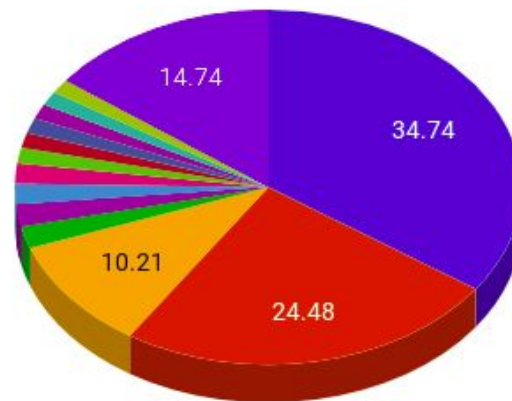
● Loop-1
 ● Loop-2
 ● Loop-3
 ● Loop-4
 ● Loop-5
 ● Loop-6
 ● Loop-7
 ● Loop-8
 ● Loop-9
 ● Loop-10
 ● Loop-11
 ● Loop-(12-2877)

Result Analysis: SPEC CPU 2006 (FP)

Povray (SPEC 2006 FP)



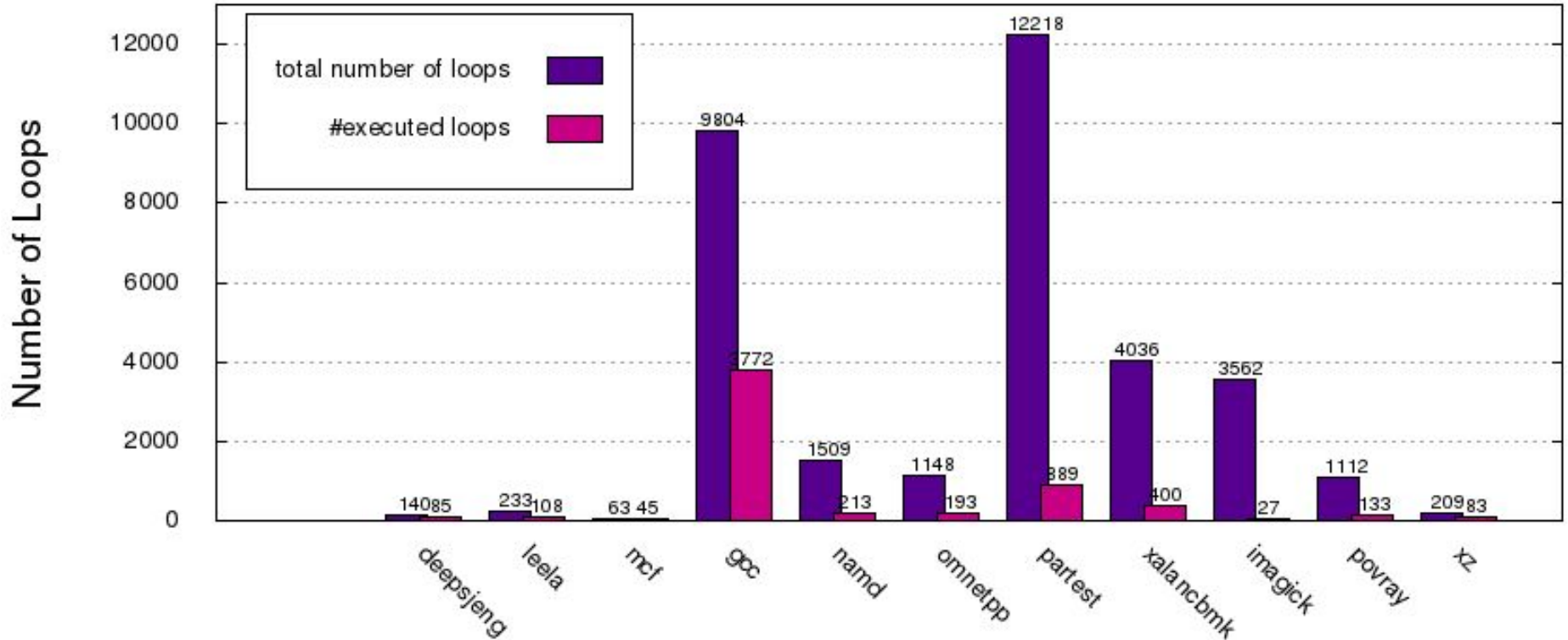
namd (SPEC 2006 FP)



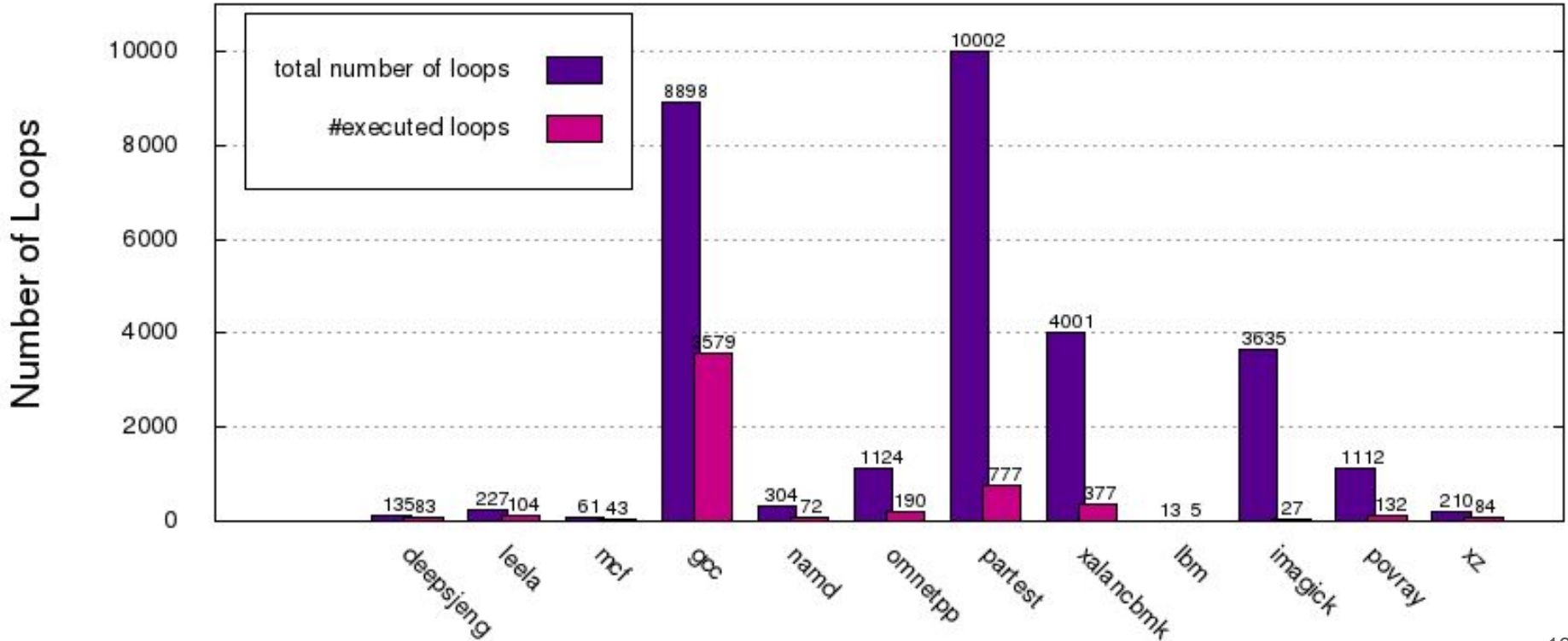
Loop-1 Loop-2 Loop-3 Loop-4 Loop-5 Loop-6 Loop-7
Loop-8 Loop-9 Loop-10 Loop-(11-1108)

Loop-1 Loop-2 Loop-3 Loop-4 Loop-5 Loop-6 Loop-7
Loop-8 Loop-9 Loop-10 Loop-11 Loop-12 Loop-13 Loop-(14-575)

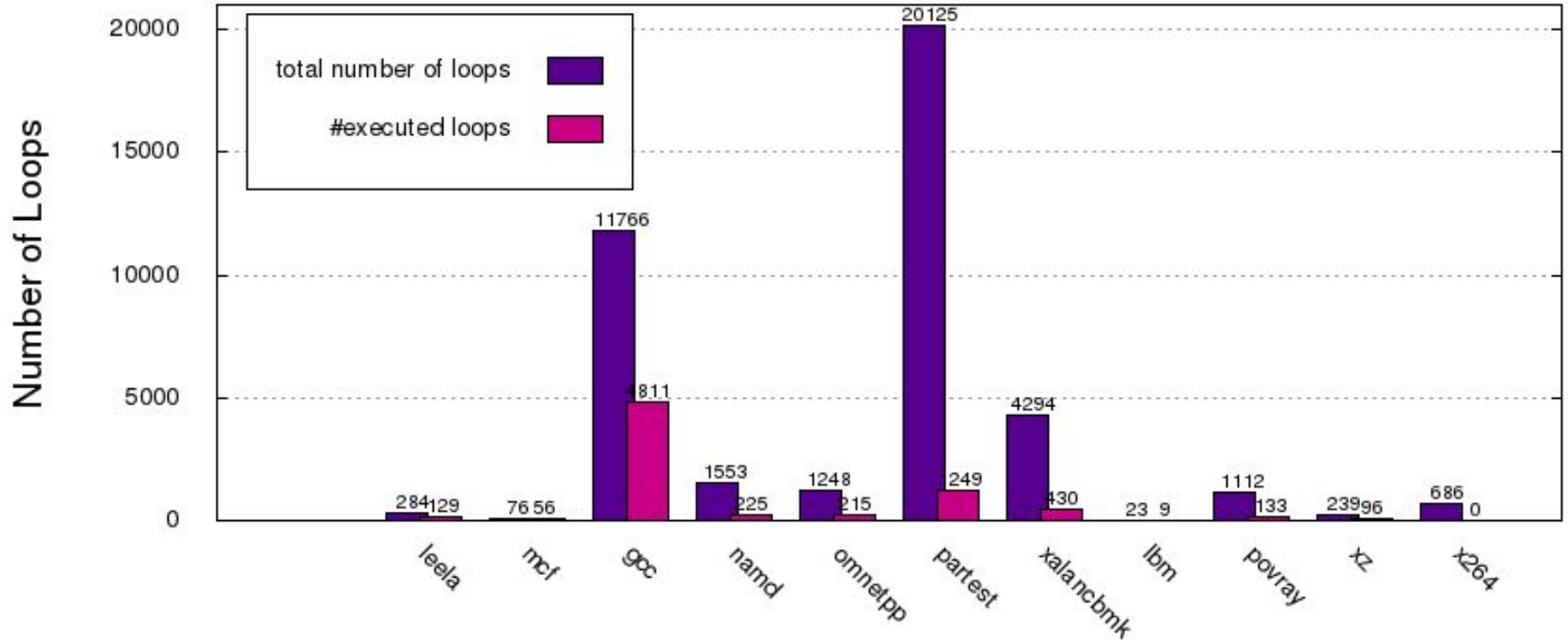
Results: SPEC CPU 2017 (Inner Loop)



Results: SPEC CPU 2017 (Outer Loop)

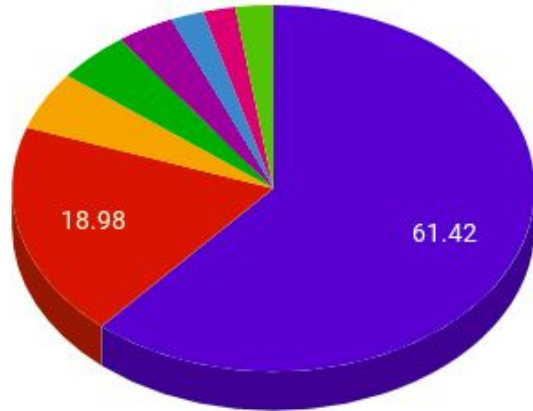


Results: SPEC CPU 2017 (All Loops)

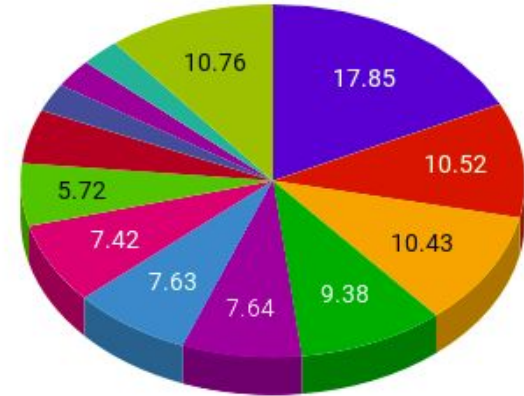


Result Analysis: SPEC CPU 2017 (INT)

omnetpp (SPEC CPU 2017 INT)



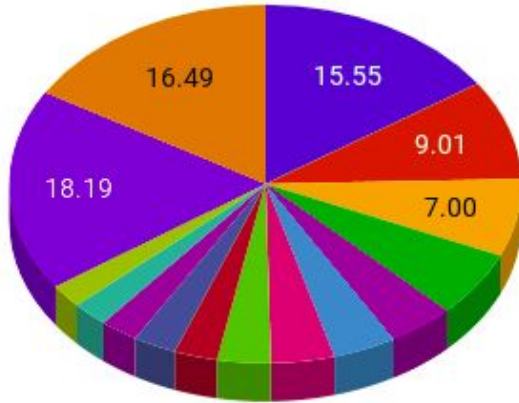
xalancbmk (SPEC CPU 2017 INT)



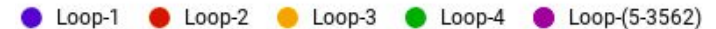
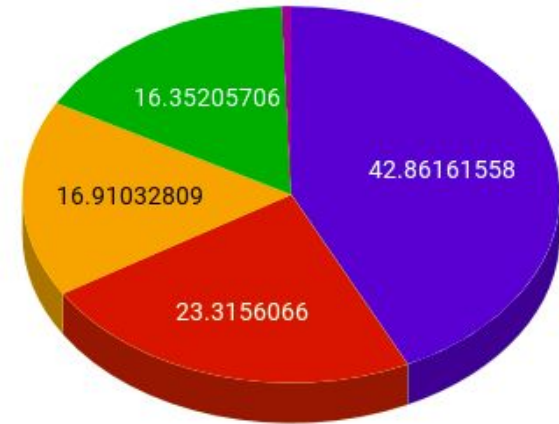
- Loop-1
- Loop-2
- Loop-3
- Loop-4
- Loop-5
- Loop-6
- Loop-7
- Loop-8
- Loop-9
- Loop-10
- Loop-11
- Loop-12
- Loop-13-4036
- Loop-8-1148

Result Analysis: SPEC CPU 2017 (FP)

partst (SPEC CPU 2017 FP)



imagick(SPEC CPU 2017 FP)



Thank You!