2024 - NS - 857-6F2 - ASMprofiler - harve106@purdue.edu - Kyle Harvey

CERIAS

The Center for Education and Research in Information Assurance and Security

ASMprofiler

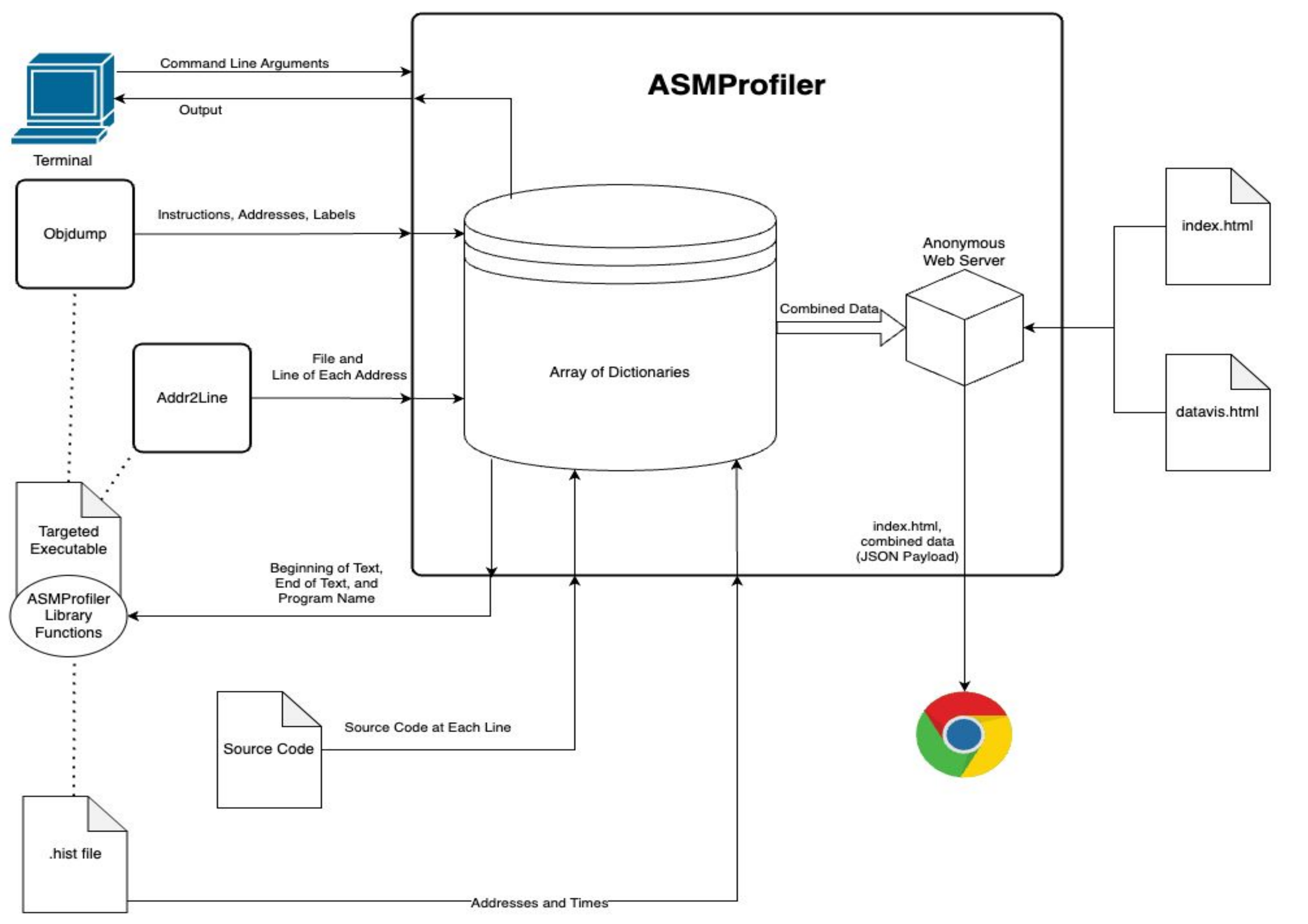
Kyle Harvey,
Nicholas Bogan,
Professor Gustavo Rodriguez-Rivera

Overview

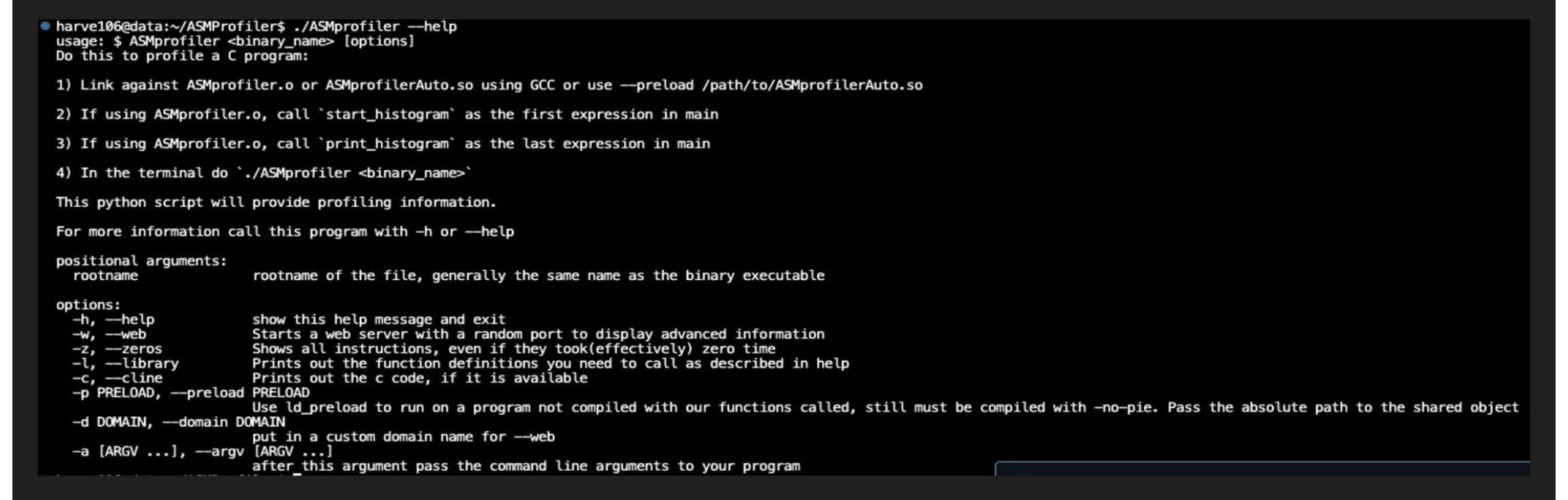
- Accept command line arguments which dictate how to run
- Fetch data from the .text section of the binary using **objdump**
- Execute binary with the addition of a function call before and after main that generates a histogram file
  - Main hook in shared library
  - Manual function calls with object file
- Combine histogram with data from **addr2line,** source code, and data collected before execution by **objdump**
- Display data as an easy to understand web page with enhanced functionality, or as simple terminal output

Motivation

- Existing profilers point to bottlenecks in C code and higher level languages
- Few profilers exist at the level of Assembly language
- Many programmers would like to find out which instructions take the most time, even if the instructions are generated from C code by compilers
- Many programmers would like to also use the profiler for programs written in Assembly code
- Many programmers would like to view the output in a user friendly manner that makes clear the association between Assembly code and higher level programming constructs
- Many programmers would also like to run the program at full speed without having to instrument the code



Python3 Script Options

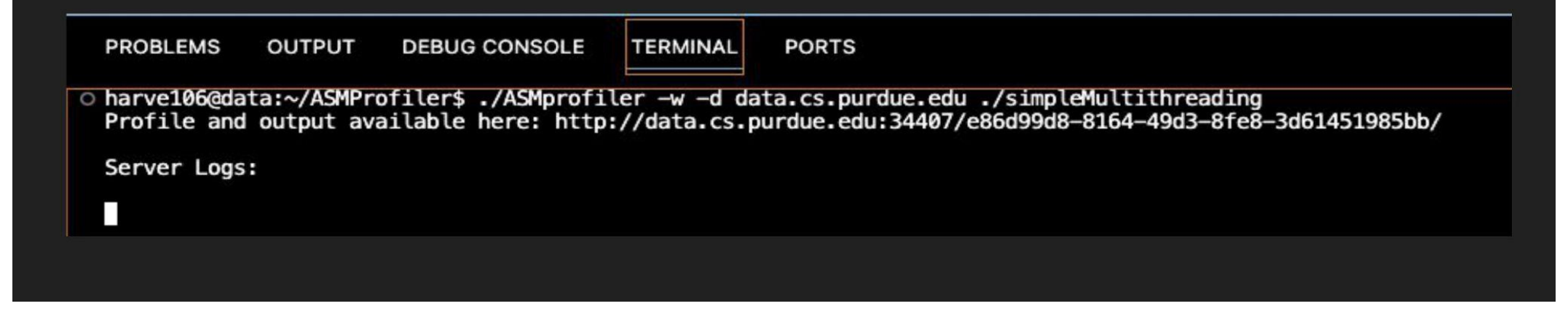


```
harve106@data:~/ASMProfiler$ ./ASMprofiler --help
usage: $ ASMprofiler <binary_name> [options]
Do this to profile a C program:

1) Link against ASMprofiler.o or ASMprofilerAuto.so using GCC or use --preload /path/to/ASMprofilerAuto.so

2) If using ASMprofiler.o, call `start_histogram` as the first expression in main

3) If using ASMprofiler.o, call `print_histogram` as the last expression in main

4) In the terminal do `./ASMprofiler <binary_name>`

This python script will provide profiling information.

For more information call this program with -h or --help

positional arguments:
  rootname              rootname of the file, generally the same name as the binary executable

options:
  -h, --help            show this help message and exit
  -w, --web             Starts a web server with a random port to display advanced information
  -z, --zeros           Shows all instructions, even if they took(effectively) zero time
  -l, --library         Prints out the function definitions you need to call as described in help
  -c, --cline           Prints out the c code, if it is available
  -p PRELOAD, --preload PRELOAD
                        Use ld_preload to run on a program not compiled with our functions called, still must be compiled with -no-pie. Pass the absolute path to the shared object
  -d DOMAIN, --domain DOMAIN
                        put in a custom domain name for --web
  -a [ARGV ...], --argv [ARGV ...]
                        after this argument pass the command line arguments to your program
```

Python3 Script Options (cont'd)

--domain DOMAIN

- This option is added to show a domain name that is not localhost when presenting the user with a link to click on for --web
- Useful when SSH'd into a remote server such as data



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

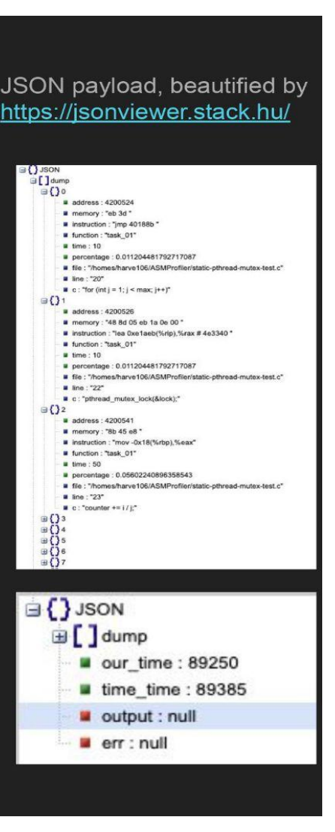
```
harve106@data:~/ASMProfiler$ ./ASMprofiler -w -d data.cs.purdue.edu ./simpleMultithreading
Profile and output available here: http://data.cs.purdue.edu:34407/e86d99d8-8164-49d3-8fe8-3d61451985bb/

Server Logs:
```

Python3 Script Options (cont'd)

JSON payload, beautified by https://jsonviewer.stack.hu/

--web

- Hosts an anonymous web page on all available interfaces
- Uses a uuid in the URL as well as a random port to ensure no collisions or inadvertent data exposure if ran on shared machine
- Hosts a webpage that requests a JSON payload from local machine
  - JSON payload is available to be used by other applications and is user viewable by appending '1json' onto the URL of the webpage
- This option automatically enables --cline in order to get source code information if available



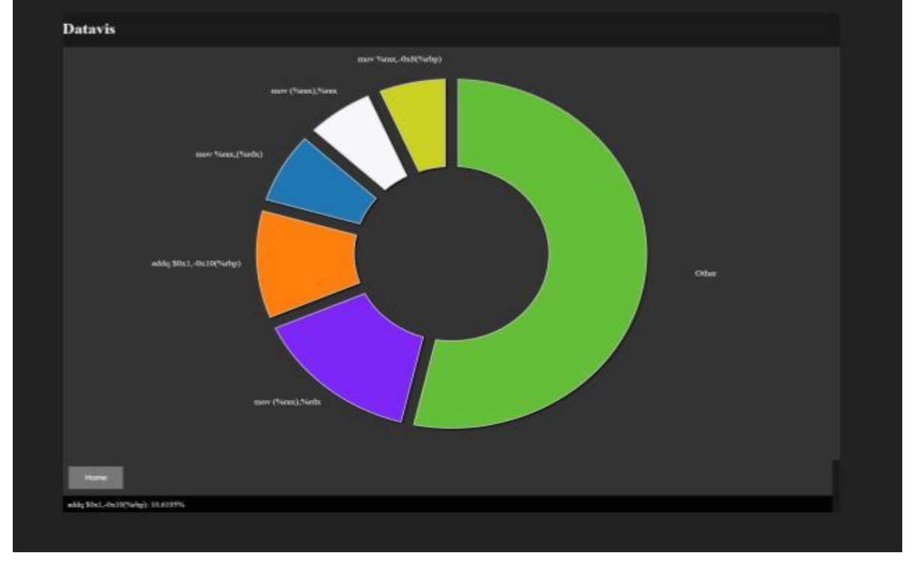
```
{} JSON
  [] dump
  out_time : 89250
  time_time : 89385
  output : null
  err : null
```

Python3 Script Options (cont'd)

--preload PRELOAD

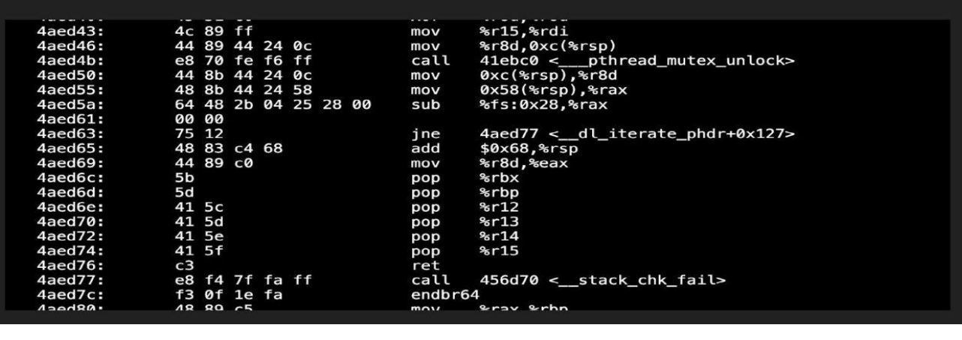
- This option is added to attempt to use LD_PRELOAD to load the shared library with any program compiled without our library
- The shared library simply gets a main hook, and calls **start_histogram()** before main and **print_histogram()** after main, generating a .hist file to be consumed by the Python3 script
- Main hook skeleton code found at https://gist.github.com/apsun/1e144bf7639b22ff0097171fa0f8c6b1

Display Data(Datavis)



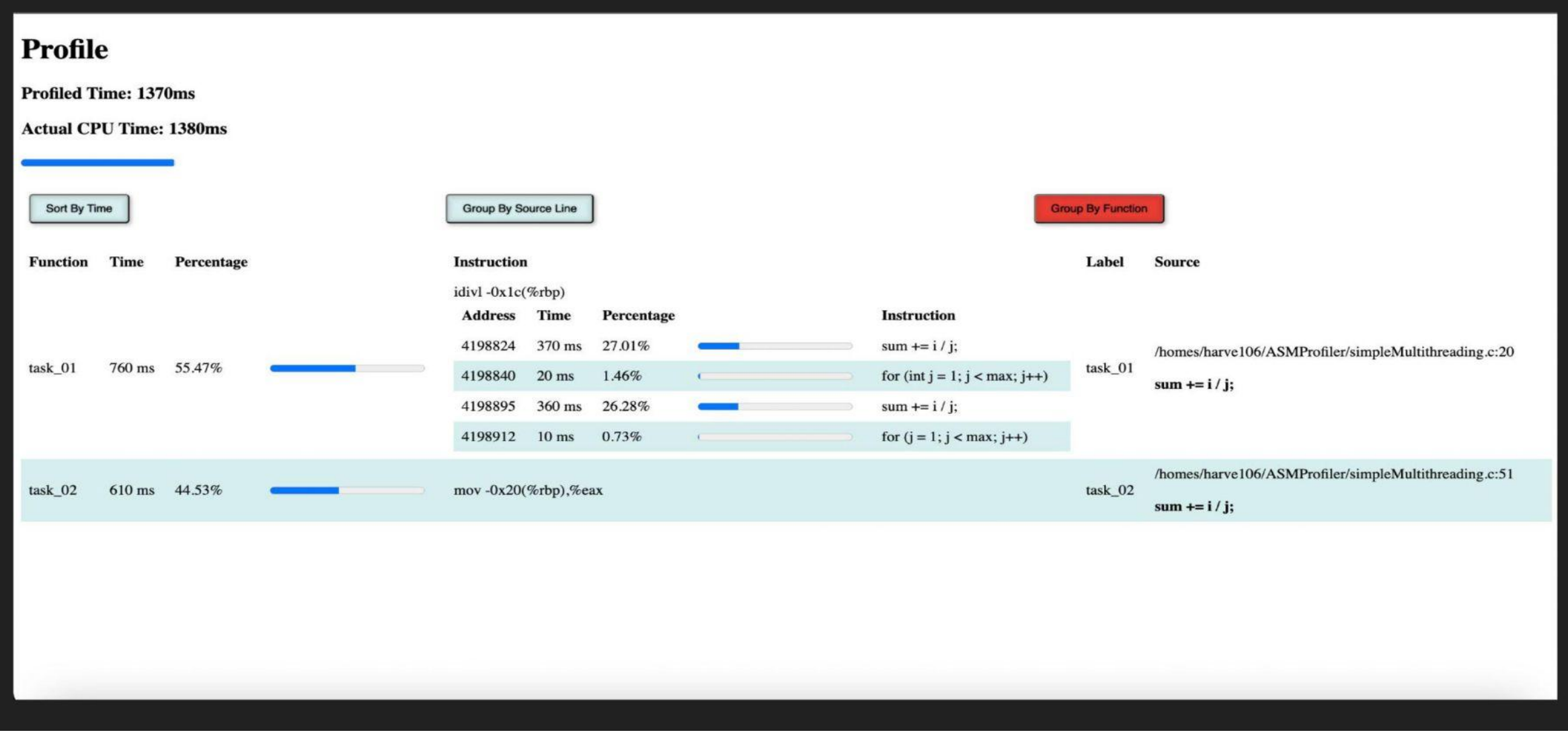
objdump Usage Before Execution

- objdump is used to get all of the bytecode, ASM instructions, function/label names and addresses inside of the .text section of the binary
- This is stored in a python dictionary for use in later
- The Python3 program passes the beginning and ending addresses of the .text section to our C library as environment variables when it calls the binary as a subprocess



```
4aed43:   4c 89 ff                 mov    %r15,%rdi
4aed46:   44 89 44 24 0c           mov    %r8d,0xc(%rsp)
4aed4b:   e8 70 fe f6 ff           call   41ebc0 <___pthread_mutex_unlock>
4aed50:   44 8b 44 24 0c           mov    0xc(%rsp),%r8d
4aed55:   48 8b 44 24 58           mov    0x58(%rsp),%rax
4aed5a:   64 48 2b 04 25 28 00     sub    %fs:0x28,%rax
4aed61:   00 00
4aed63:   75 12                    jne    4aed77 <__dl_iterate_phdr+0x127>
4aed65:   48 83 c4 68              add    $0x68,%rsp
4aed69:   44 89 c0                 mov    %r8d,%eax
4aed6c:   5b                       pop    %rbx
4aed6d:   5d                       pop    %rbp
4aed6e:   41 5c                    pop    %r12
4aed70:   41 5d                    pop    %r13
4aed72:   41 5e                    pop    %r14
4aed74:   41 5f                    pop    %r15
4aed76:   c3                       ret
4aed77:   e8 f4 7f fa ff           call   456d70 <__stack_chk_fail>
4aed7c:   f3 0f 1e fa              endbr64
```

Display Data(Grouping by Function Functionality)



Profile

Profiled Time: 1370ms

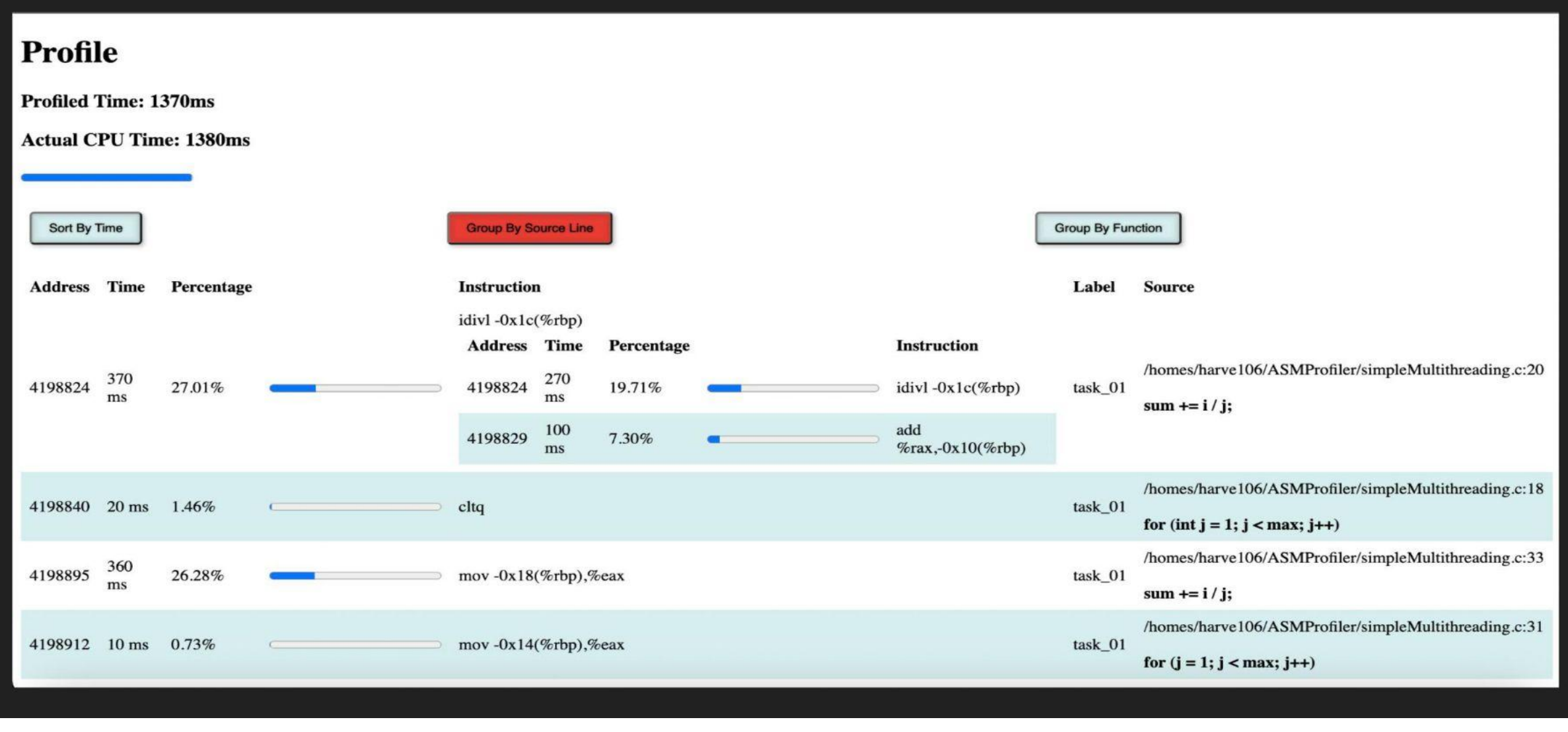
Actual CPU Time: 1380ms

Sort By Time | Group By Source Line | Group By Function

Function	Time	Percentage	Instruction	Label	Source
task_01	760 ms	55.47%	idivl -0x1c(%rbp)	task_01	/homes/harve106/ASMProfiler/simpleMultithreading.c:20 **sum += i / j;**
task_02	610 ms	44.53%	mov -0x20(%rbp),%eax	task_02	/homes/harve106/ASMProfiler/simpleMultithreading.c:51 **sum += i / j;**

Address	Time	Percentage	Instruction
4198824	370 ms	27.01%	sum += i / j;
4198840	20 ms	1.46%	for (int j = 1; j < max; j++)
4198895	360 ms	26.28%	sum += i / j;
4198912	10 ms	0.73%	for (j = 1; j < max; j++)

Display Data(Grouping by C Source Line Functionality)



Profile

Profiled Time: 1370ms

Actual CPU Time: 1380ms

Sort By Time | Group By Source Line | Group By Function

Address	Time	Percentage	Instruction	Label	Source
4198824	370 ms	27.01%	idivl -0x1c(%rbp)	task_01	/homes/harve106/ASMProfiler/simpleMultithreading.c:20 **sum += i / j;**
4198840	20 ms	1.46%	cltq	task_01	/homes/harve106/ASMProfiler/simpleMultithreading.c:18 **for (int j = 1; j < max; j++)**
4198895	360 ms	26.28%	mov -0x18(%rbp),%eax	task_01	/homes/harve106/ASMProfiler/simpleMultithreading.c:33 **sum += i / j;**
4198912	10 ms	0.73%	mov -0x14(%rbp),%eax	task_01	/homes/harve106/ASMProfiler/simpleMultithreading.c:31 **for (j = 1; j < max; j++)**

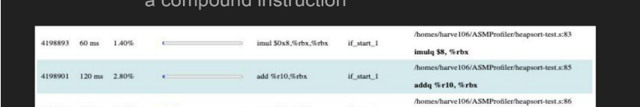
Address	Time	Percentage	Instruction
4198824	270 ms	19.71%	idivl -0x1c(%rbp)
4198829	100 ms	7.30%	add %rax,-0x10(%rbp)

Binary Execution

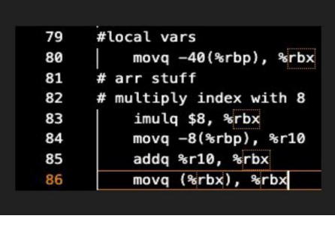
- The binary is executed from the Python3 script as a subprocess, passing the beginning and ending addresses of the .text section as well as other data using environment variables
- **start_histogram()** is called before or at the very beginning of main
  - Fetches the environment variables
  - Initializes histogram array
  - Calls **profil()**
- **print_histogram()**
  - Called after or at the very end of main
  - Prints the histogram array data to a .hist file
- after execution
  - Data from POPEN is used to calculate the total cpu time to compare to the total time in the profil histogram so the user can determine if they are missing large amounts of execution time due to dynamic library usage

Example Usage

- The instructions taking the most time are examined and are considered for optimizations
  - The top time taking instruction is an array deref, and if the instructions around it are viewed, they take a substantial amount of time as well
  - If lines 83-86 can be optimized to under 920ms, time will have been saved
  - In this case, the optimization will be the replacement of this and other similar instructions with a compound instruction



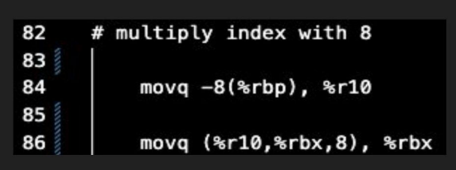
4198893	60 ms	1.40%	imul $0x8,%rbx,%rbx	if_start_1	/homes/harve106/ASMProfiler/heapsort-test.s:83 **imulq $8, %rbx**
4198901	120 ms	2.80%	add %r10,%rbx	if_start_1	/homes/harve106/ASMProfiler/heapsort-test.s:85 **addq %r10, %rbx**
4198904	740 ms	17.29%	mov (%rbx),%rbx	if_start_1	/homes/harve106/ASMProfiler/heapsort-test.s:86 **movq (%rbx), %rbx**



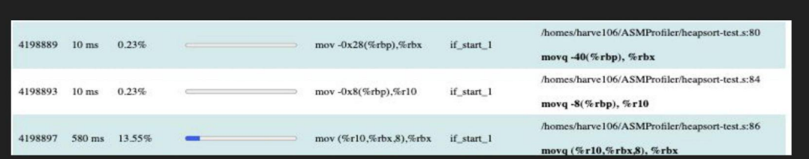
```
79  #local vars
80      movq -40(%rbp), %rbx
81  # arr stuff
82  # multiply index with 8
83      imulq $8, %rbx
84      movq -8(%rbp), %r10
85      addq %r10, %rbx
86      movq (%rbx), %rbx
```

Example Usage

- Success
- This optimization saved 340 ms



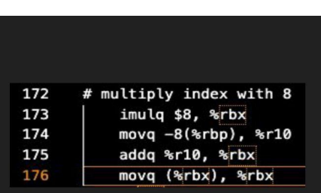
```
82  # multiply index with 8
83
84      movq -8(%rbp), %r10
85
86      movq (%r10,%rbx,8), %rbx
```



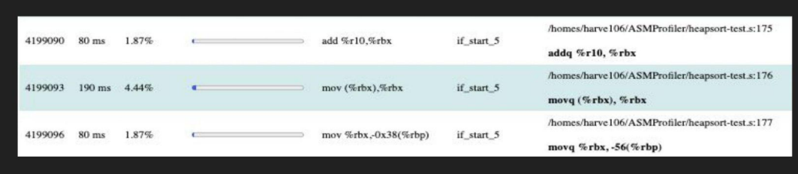
4198889	10 ms	0.23%	mov -0x28(%rbp),%rbx	if_start_1	/homes/harve106/ASMProfiler/heapsort-test.s:80 **movq -40(%rbp), %rbx**
4198893	10 ms	0.23%	mov -0x8(%rbp),%r10	if_start_1	/homes/harve106/ASMProfiler/heapsort-test.s:84 **movq -8(%rbp), %r10**
4198897	580 ms	13.55%	mov (%r10,%rbx,8),%rbx	if_start_1	/homes/harve106/ASMProfiler/heapsort-test.s:86 **movq (%r10,%rbx,8), %rbx**

Example Usage

- A similar case at 175-177
- Time to beat for this operation is 350 ms



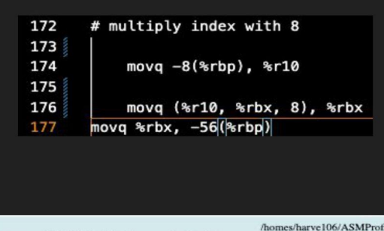
```
172  # multiply index with 8
173      imulq $8, %rbx
174      movq -8(%rbp), %r10
175      addq %r10, %rbx
176      movq (%rbx), %rbx
```



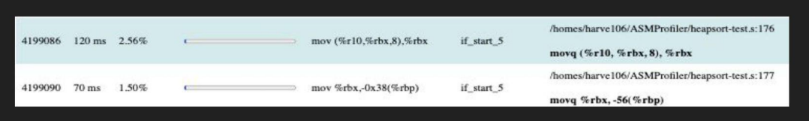
4199090	80 ms	1.87%	add %r10,%rbx	if_start_5	/homes/harve106/ASMProfiler/heapsort-test.s:175 **addq %r10, %rbx**
4199093	190 ms	4.44%	mov (%rbx),%rbx	if_start_5	/homes/harve106/ASMProfiler/heapsort-test.s:176 **movq (%rbx), %rbx**
4199096	80 ms	1.87%	mov %rbx,-0x38(%rbp)	if_start_5	/homes/harve106/ASMProfiler/heapsort-test.s:177 **movq %rbx, -56(%rbp)**

Example Usage

- Success



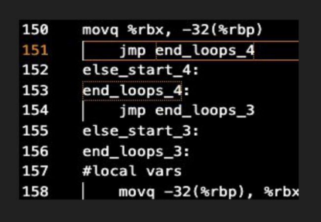
```
172  # multiply index with 8
173
174      movq -8(%rbp), %r10
175
176      movq (%r10, %rbx, 8), %rbx
177  movq %rbx, -56(%rbp)
```



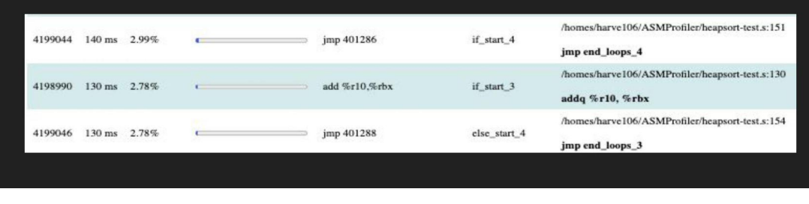
4199086	120 ms	2.56%	mov (%r10,%rbx,8),%rbx	if_start_5	/homes/harve106/ASMProfiler/heapsort-test.s:176 **movq (%r10, %rbx, 8), %rbx**
4199090	70 ms	1.50%	mov %rbx,-0x38(%rbp)	if_start_5	/homes/harve106/ASMProfiler/heapsort-test.s:177 **movq %rbx, -56(%rbp)**

Example Usage

- Examination of unconditional jumps
- Instructions are called frequently
- If possible an unconditional jump could be:
  - Inlined
  - Simplified
  - Or eliminated



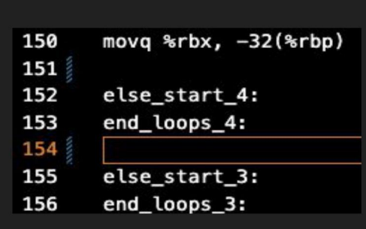
```
150  movq %rbx, -32(%rbp)
151      jmp end_loops_4
152  else_start_4:
153  end_loops_4:
154      jmp end_loops_3
155  else_start_3:
156  end_loops_3:
157  #local vars
158      movq -32(%rbp), %rbx
```



4199044	140 ms	2.99%	jmp 401286	if_start_4	/homes/harve106/ASMProfiler/heapsort-test.s:151 **jmp end_loops_4**
4198990	130 ms	2.78%	add %r10,%rbx	if_start_3	/homes/harve106/ASMProfiler/heapsort-test.s:130 **addq %r10, %rbx**
4199046	130 ms	2.78%	jmp 401288	else_start_4	/homes/harve106/ASMProfiler/heapsort-test.s:154 **jmp end_loops_3**

Example Usage

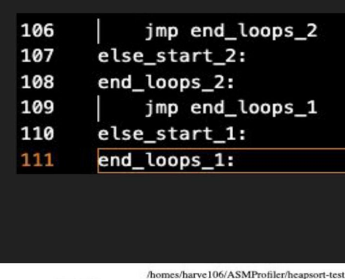
- These can be eliminated



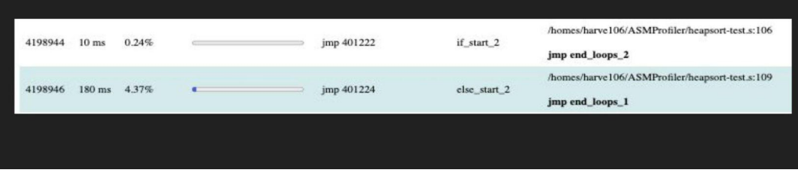
```
150  movq %rbx, -32(%rbp)
151
152  else_start_4:
153  end_loops_4:
154
155  else_start_3:
156  end_loops_3:
```

Example Usage

- Similar case at 106, 109



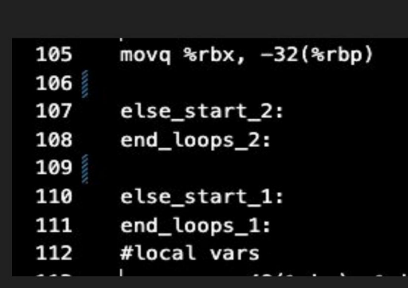
```
106      jmp end_loops_2
107  else_start_2:
108  end_loops_2:
109      jmp end_loops_1
110  else_start_1:
111  end_loops_1:
```



4198944	10 ms	0.24%	jmp 401222	if_start_2	/homes/harve106/ASMProfiler/heapsort-test.s:106 **jmp end_loops_2**
4198946	180 ms	4.37%	jmp 401224	else_start_2	/homes/harve106/ASMProfiler/heapsort-test.s:109 **jmp end_loops_1**

Example Usage

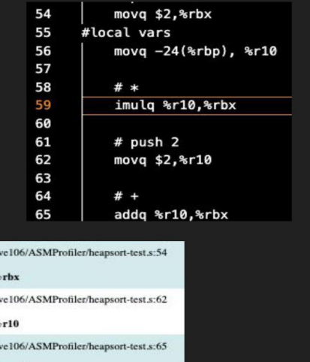
- Gone



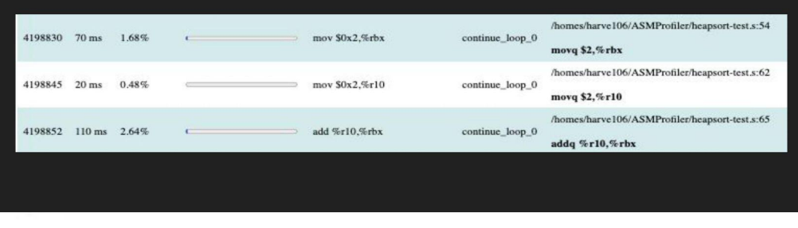
```
105  movq %rbx, -32(%rbp)
106
107  else_start_2:
108  end_loops_2:
109
110  else_start_1:
111  end_loops_1:
112  #local vars
```

Example Usage

- Arithmetic operations can often be simplified
- In this case the program is doing 2 * %r10 + 2 with the result ending up in %rbx
- This can be simplified using lea instruction because the multiplication is done by two



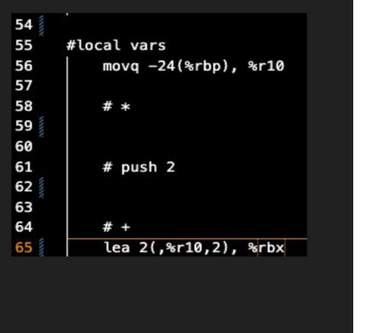
```
54      movq $2,%rbx
55  #local vars
56      movq -24(%rbp), %r10
57
58      # *
59      imulq %r10,%rbx
60
61      # push 2
62      movq $2,%r10
63
64      # +
65      addq %r10,%rbx
```



4198830	70 ms	1.68%	mov $0x2,%rbx	continue_loop_0	/homes/harve106/ASMProfiler/heapsort-test.s:54 **movq $2,%rbx**
4198845	20 ms	0.48%	mov $0x2,%r10	continue_loop_0	/homes/harve106/ASMProfiler/heapsort-test.s:62 **movq $2,%r10**
4198852	110 ms	2.64%	add %r10,%rbx	continue_loop_0	/homes/harve106/ASMProfiler/heapsort-test.s:65 **addq %r10,%rbx**

Example Usage

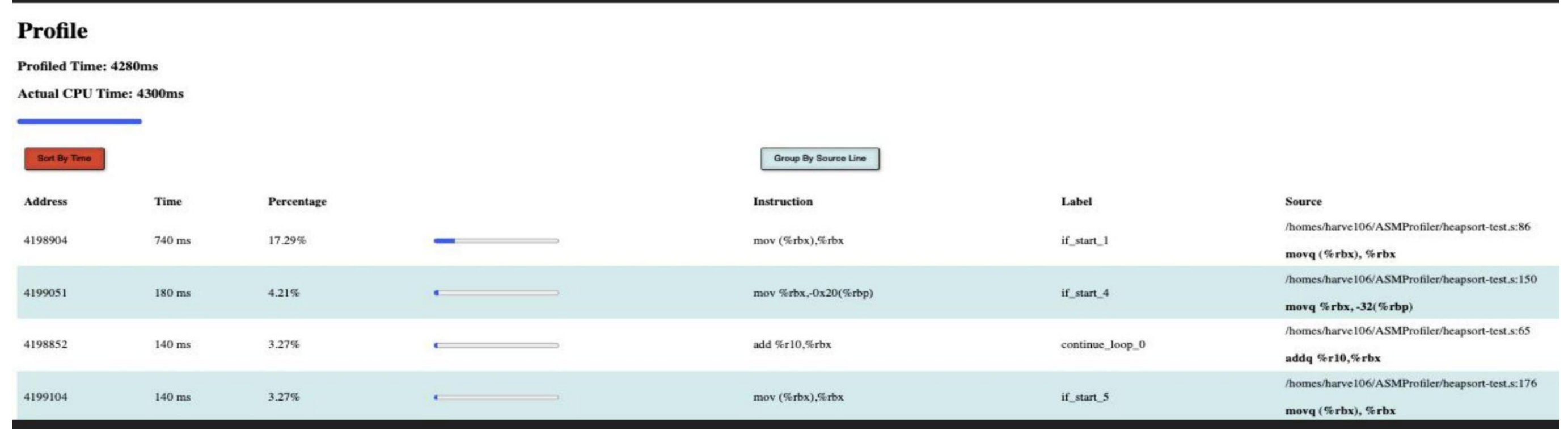
- This simplification is fast enough that line 65 no longer appears in the profile



```
54
55  #local vars
56      movq -24(%rbp), %r10
57
58      # *
59
60
61      # push 2
62
63
64      # +
65      lea 2(,%r10,2), %rbx
```

Example Usage

- First, the program is ran with the profiler and the list of top time taking instructions is analyzed



Profile

Profiled Time: 4280ms

Actual CPU Time: 4300ms

Sort By Time | Group By Source Line

Address	Time	Percentage	Instruction	Label	Source
4198904	740 ms	17.29%	mov (%rbx),%rbx	if_start_1	/homes/harve106/ASMProfiler/heapsort-test.s:86 **movq (%rbx), %rbx**
4199051	180 ms	4.21%	mov %rbx,-0x20(%rbp)	if_start_4	/homes/harve106/ASMProfiler/heapsort-test.s:150 **movq %rbx, -32(%rbp)**
4198852	140 ms	3.27%	add %r10,%rbx	continue_loop_0	/homes/harve106/ASMProfiler/heapsort-test.s:65 **addq %r10,%rbx**
4199104	140 ms	3.27%	mov (%rbx),%rbx	if_start_5	/homes/harve106/ASMProfiler/heapsort-test.s:176 **movq (%rbx), %rbx**

Example Usage

- 1000 runs of the optimized and unoptimized version were ran and the average was taken
- Unoptimized: 4.292475
- Optimized: 4.08495

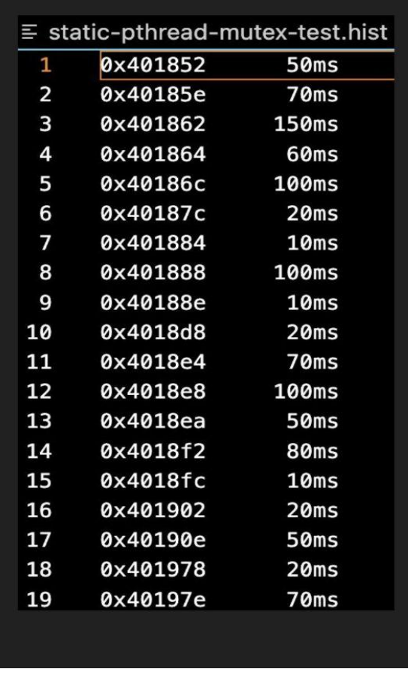
Conclusion

- ASMprofiler is usable to optimize assembly and C programs
- ASMprofiler was used in computer architecture class by over 500 students to optimize a hash table dictionary
  - Students found that most of the time of the hash function was spent in the modulo operation
  - Was optimized by approximating the modulo by shift operations and subtractions using a prime number close to a power of two

Combine Histogram with objdump Data

- Because x86_64 Assembly is CISC, each two byte bucket profiled by **profil()** may only be a part of a single assembly instruction
- Summing multiple lines of the histogram in order to get the time for one Assembly instruction is necessary
- To keep this linear time amortized, a destructive search method that treats the profile from the .hist file as a stack is used

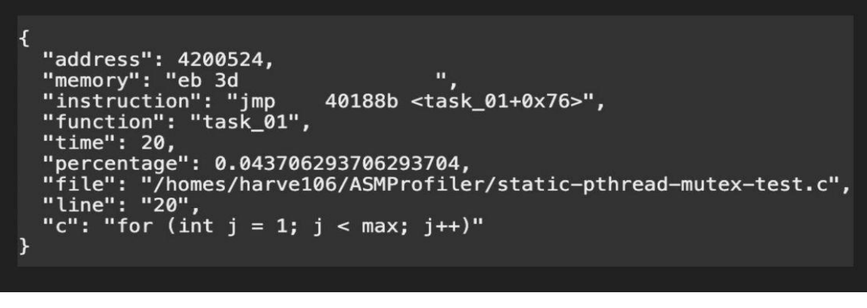
Example .hist file



```
static-pthread-mutex-test.hist
0x401852    50ms
0x40185e    70ms
0x401862   150ms
0x401864    60ms
0x40186c   100ms
0x40187c    20ms
0x401884    10ms
0x401888   100ms
0x40188e    10ms
0x4018d8    20ms
0x4018e4    70ms
0x4018e8   100ms
0x4018ea    50ms
0x4018f2    80ms
0x4018fc    10ms
0x401902    20ms
0x40190e    50ms
0x401978    20ms
0x40197e    70ms
```

Combine Existing Profile with addr2line Data and Source

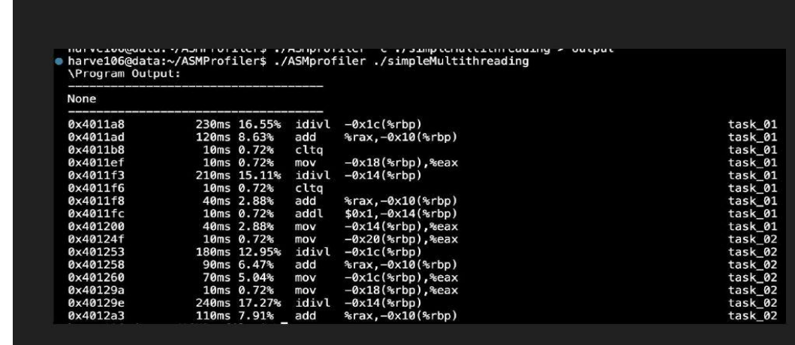
- addr2line is called with one address per assembly instruction profiled
- The data from addr2line is then parsed and the source code file is read, adding the content of each line to the profile data structure
  - A list of the files the Python3 program has already opened is maintained and read so as not to have a file open and close for each assembly instruction
- The resulting data structure has each time-taking assembly instructions' address, bytecode, ASM code, source-code line, source-code content, function, and time taken

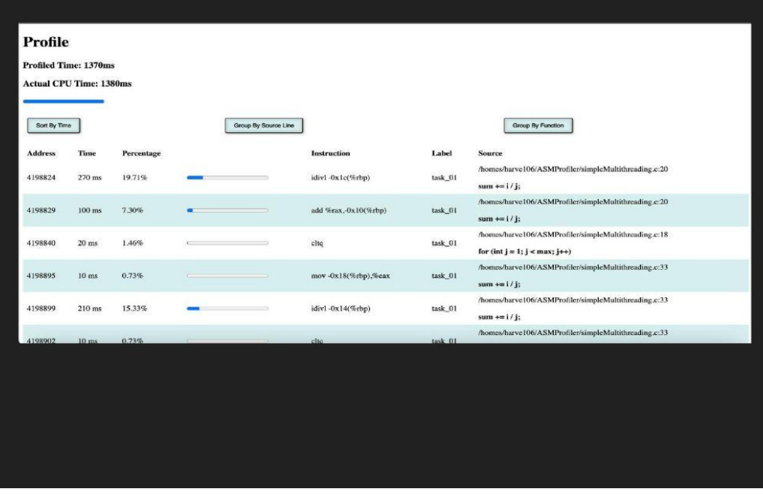


```
{
  "address": 4200524,
  "memory": "eb 3d                 ",
  "instruction": "jmp    40188b <task_01+0x76>",
  "function": "task_01",
  "time": 20,
  "percentage": 0.043706293706293704,
  "file": "/homes/harve106/ASMProfiler/static-pthread-mutex-test.c",
  "line": "20",
  "c": "for (int j = 1; j < max; j++)"
}
```

Display Data

- The data is then displayed through either the command line or the web interface





PURDUE UNIVERSITY

CERIAS