Testing your program

./sigil_test.sh will run sigil on an executable file. The variables are explained in Table 1, while the lines are explained in Table 2. The following script creates a file postprocessing_result. Table 2 shows variables that need to bechanged with an example.

Note that the script is written to support

Table 1 environmental variables that need to be set

Variable	Description
sigil_path	Where sigil has been compiled
exe_file_path	Where user's executable file (after compiling has been stored)
Postprocessing_path	Where ./aggregate_costs_gran.py is located
callgrind_output_file	Where user stores the output of callgrind

Table 2 variables to be changed by user

Variable	Description
sigil_path	Where sigil has been compiled
exe_file_path	Where user's executable file (after compiling has been stored).

Table 3 shows the commands and their explanations. It provides you the option to edit the path to each executable file.

Table 3 main commands for using sigil

Command line	Explanation
$sigil_path/vg-in-place --tool=callgrind --sigil-tool=yes --separate-callers=100 --cache-sim=yes --drw-func=yes $exe_file_path/executable_file	sigil command with necessary options, the minimum of which is 100. Cache-sim is a fag that "enables or disables collection of cache access and miss counts"[1].
$sigil_path/vg-in-place --tool=callgrind --cache-sim=yes --branch-sim=yes --callgrind-out-file=$callgrind_output_file/callgrind_out $exe_file_path/executable_file	callgrind runnable with necessary options. The out_put is renamed to callgrind_out [1].
postprocessing_path/aggregate_costs_gran.py $exe_file_path/sigil.totals.out-1 --trim-tree --cgfile=$callgrind_output_file/callgrind_out --gran-mode=metric > $callgrind_output_file/postprocessing_result	aggregate_costs_gran.py is the script that calculates aggregate costs as shown in table 3 and 4. It uses sigil and callgrind output and provides postprocessing_results.

The script will ask the use for sigil_path and user's executable file with its path. The user will have an option for default which will be displayed.

Sigil Path, points to where sigil is set up.

```
sigil_path="$sigil_path/home/DREXEL/pm626/sigil"
export sigil_path
```

Executaionable_File_Path, this parameter includes the path to your compiled file and the name of your executable file. In this example math is the executable file.

```
exe_file_path="$exe_file_path/home/DREXEL/pm626/Downloads/function/math"
export exe_file_path
```

Creates Post Processing Path, from this point the user doesn't need to change any variables.

```
postprocessing_path="$postprocessing_path $sigil_path/sigil/postprocessing"
export postprocessing_path
```

Runs Sigil

```
$sigil_path/sigil/valgrind-3.10.1/vg-in-place --tool=callgrind --sigil-tool=yes --
separate-callers=100 --cache-sim=yes --drw-func=yes $exe_file_path


$sigil_path/sigil/valgrind-3.10.1/vg-in-place --tool=callgrind --cache-sim=yes --
branch-sim=yes --callgrind-out-file=callgrind_out $exe_file_path

$postprocessing_path/aggregate_costs_gran.py sigil.totals.out-1 --trim-tree --
cgfile=callgrind_out --gran-mode=metric >postprocessing_result
```

Figure 1 shows part of post-processing output for sigil

			Function name	Numcalls	Instrs	Flops	Iops	Ipcomm_uq	Opcomm_uq	Local_uq	Ipcomm	Opcomm
396	0		main	* 1	15870	0	26786	1405	414	4783	1543	452
429	0		std::ostream::operator<<(int)	* 3	7053	0	12865	871	295	2147	947	323
432	0		std::ostream& std::ostream::_M_insert<long>(long)	* 3	6209	0	11522	779	295	1793	855	323
435	0		std::ctype<char>::_M_widen_init() const	* 1	2627	0	6250	296	35	651	308	35
436	0		std::ctype<char>::do_widen(char const*, char const	* 1	1316	0	2132	536	32	371	548	32
175	28		_dl_runtime_resolve	* 1	1190	0	1958	272	32	355	284	32
176	28		_dl_fixup	* 1	1169	0	1940	264	32	299	276	32
68	29		_dl_lookup_symbol_x	* 1	1089	0	1811	240	36	263	268	36
69	29		do_lookup_x	* 1	943	0	1612	235	12	171	239	12
38	14		_dl_name_match_p	* 5	258	0	411	63	0	16	87	0
22	47		strcmp	* 10	133	0	246	13	0	0	85	0
70	29		check_match.12439	* 1	196	0	392	67	0	38	67	0
22	48		strcmp	* 2	137	0	270	16	0	0	16	0
439	0		memcpy	* 1	113	0	165	264	0	8	264	0
441	0		std::num_put<char, std::ostreambuf_iterator<char,	* 3	2440	0	3560	385	232	624	451	260
175	29		_dl_runtime_resolve	* 1	1510	0	2284	164	71	329	166	85
176	29		_dl_fixup	* 1	1489	0	2266	156	68	273	158	76
68	30		_dl_lookup_symbol_x	* 1	1409	0	2137	132	64	237	150	64
69	30		do_lookup_x	* 1	408	0	703	127	40	133	129	40
70	30		check_match.12439	* 1	137	0	278	48	0	33	48	0
22	49		strcmp	* 1	86	0	170	8	0	0	8	0
444	0		std::ostreambuf_iterator<char, std::char_traits<ch	* 3	896	0	1232	223	164	292	293	188
413	1		__gnu_cxx::stdio_sync_filebuf<char, std::char_trai	* 3	458	0	620	160	91	104	164	91
416	1		fwrite	* 3	440	0	617	136	91	104	140	91
417	1		_IO_file_xsputn@@GLIBC_2.2.5	* 3	245	0	302	48	43	48	68	43
448	0		0x00000000000076d0	* 3	75	0	105	24	4	0	24	8
445	0		0x0000000000080920	* 3	87	0	81	48	0	42	128	0
248	14		std::locale::id::_M_id() const	* 3	15	0	18	16	0	0	48	0
175	27		_dl_runtime_resolve	* 1	887	0	1395	163	73	338	163	75
176	27		_dl_fixup	* 1	866	0	1377	155	73	282	155	75

Figure 1. partial sigil post processing output for simple_math

Note the red rectangle signifies numbers that represent either operating system functions or those that are not included in binary. Table 4 shows a brief description of these results.

Table 4. Sigil post processing output

Numcalls	number of calls to that node
Instructions	number of instructions executed in that node
Flops	Total Floating Point operations executed in that node
Iops	Total Integer operations executed in that node
Ipcomm_uq	The unique input communication seen by that node (inclusive).
Opcomm_uq	The unique output communication seen by that node (inclusive).
Local_uq	The unique local communication seen by that node (inclusive).
Ipcomm	The total input communication seen by that node (inclusive).
Opcomm	The total output communication seen by that node (inclusive).
Local	- The total local communication seen by that node (inclusive).
Comp_Comm_uniq	The ratio of computation to unique communication. This represents the work done per byte of true input data transferred. Higher ratios are more indicative of a hotspot.