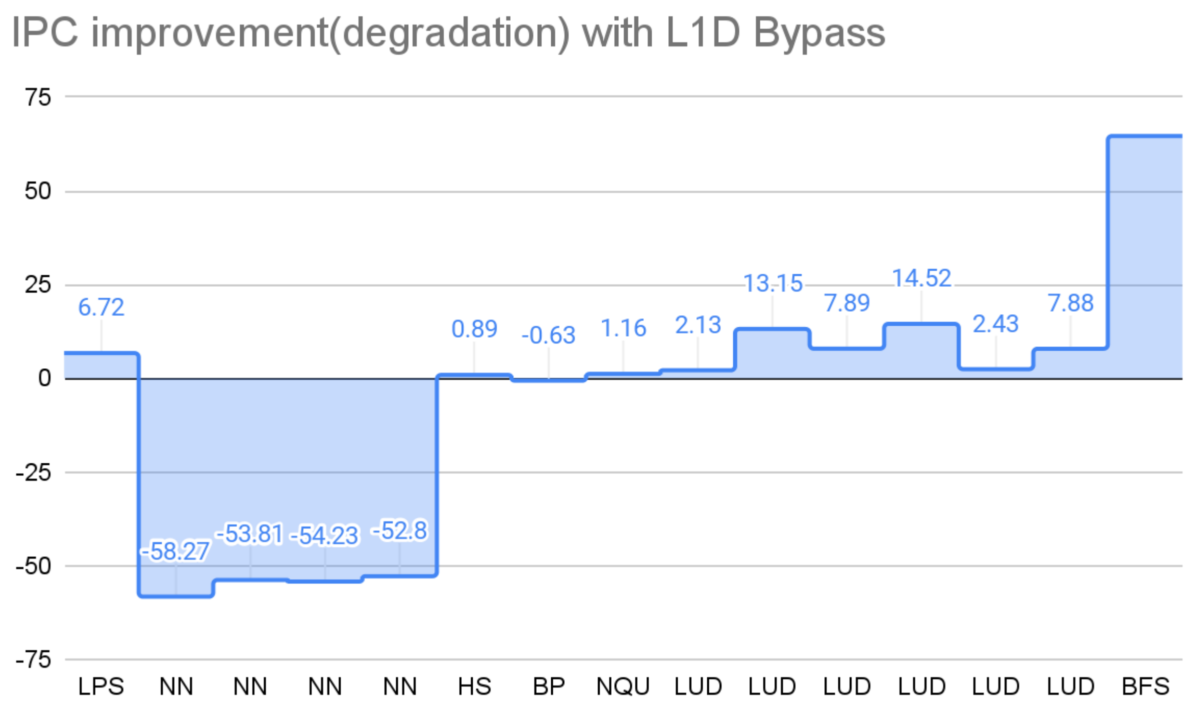
Report: Final Project
ECE786

Shawn Salekin

Task1



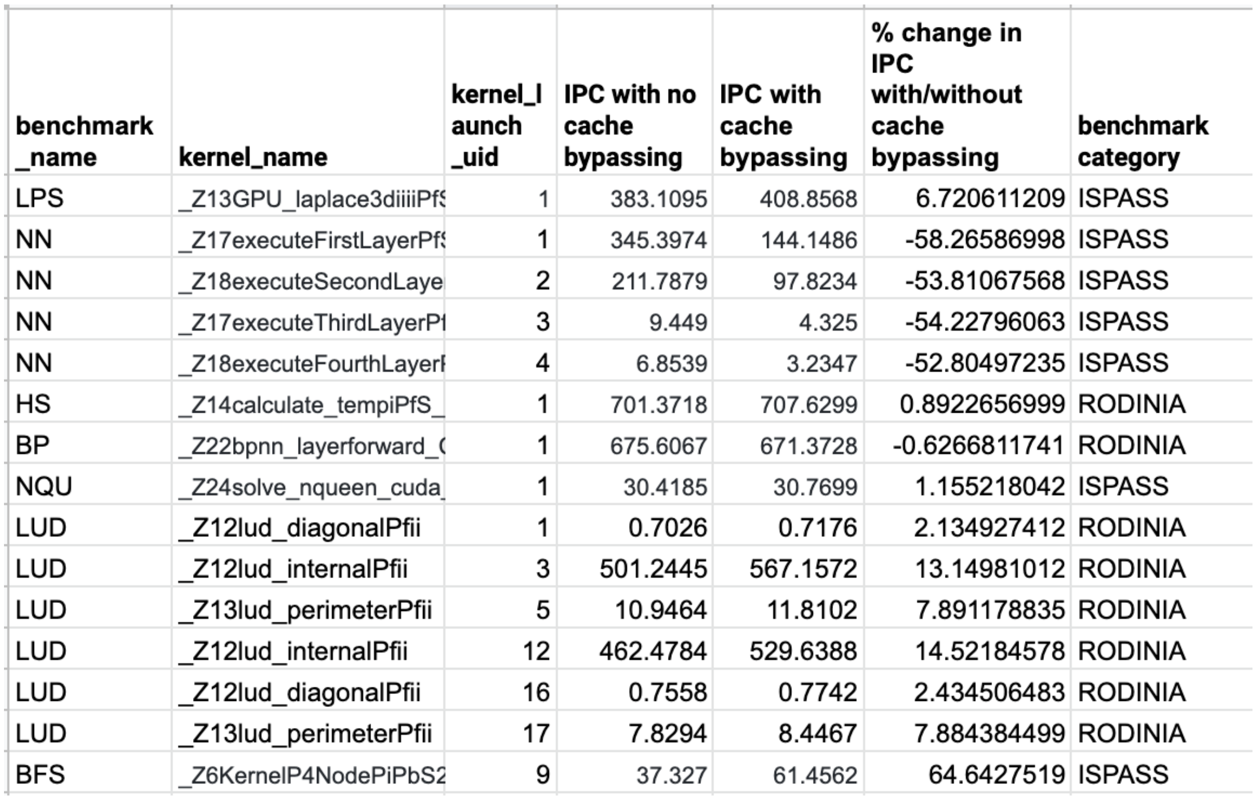
On the chart above, the vertical axis indicates the percentage improvement(above 0) or degradation(below 0) of IPC when L1D bypass was turned on. The ones at the bottom of the graph thus can be marked as cache-friendly, while the ones in the top half of the graph can be called cache-unfriendly.

The ones that are close to the 0 (BP, HS) can be called cache-insensitive since their IPCs do not change much regardless of whether bypassing was turned on or off.

To summarize

1. Cache Unfreindly (>1.00): BFS, LPS, NQU, LUD,
2. Cache Insensitive(close to 0): BP, HS
3. Cache Friendly (<-1.00): NN

Following is the table created from the data collected for task1.



benchmark_name	kernel_name	kernel_launch_uid	IPC with no cache bypassing	IPC with cache bypassing	% change in IPC with/without cache bypassing	benchmark category
LPS	_Z13GPU_laplace3diiiiPfS	1	383.1095	408.8568	6.720611209	ISPASS
NN	_Z17executeFirstLayerPfS	1	345.3974	144.1486	-58.26586998	ISPASS
NN	_Z18executeSecondLaye	2	211.7879	97.8234	-53.81067568	ISPASS
NN	_Z17executeThirdLayerPf	3	9.449	4.325	-54.22796063	ISPASS
NN	_Z18executeFourthLayerI	4	6.8539	3.2347	-52.80497235	ISPASS
HS	_Z14calculate_tempiPfS_	1	701.3718	707.6299	0.8922656999	RODINIA
BP	_Z22bpnn_layerforward_(	1	675.6067	671.3728	-0.6266811741	RODINIA
NQU	_Z24solve_nqueen_cuda_	1	30.4185	30.7699	1.155218042	ISPASS
LUD	_Z12lud_diagonalPfii	1	0.7026	0.7176	2.134927412	RODINIA
LUD	_Z12lud_internalPfii	3	501.2445	567.1572	13.14981012	RODINIA
LUD	_Z13lud_perimeterPfii	5	10.9464	11.8102	7.891178835	RODINIA
LUD	_Z12lud_internalPfii	12	462.4784	529.6388	14.52184578	RODINIA
LUD	_Z12lud_diagonalPfii	16	0.7558	0.7742	2.434506483	RODINIA
LUD	_Z13lud_perimeterPfii	17	7.8294	8.4467	7.884384499	RODINIA
BFS	_Z6KernelP4NodePiPbS2	9	37.327	61.4562	64.6427519	ISPASS

Task2

How To Run

To collect a cache profile, create a file called `` `profile.txt` `` in the directory you will run the benchmark on with the following content:

```
collect 1
load 0
use 0
```

This will set the parameters so that a file called `` `refcount.txt` `` will be created where all the address references will be collected.

To use the references for cache bypassing, change the content of profile.txt like the following:

```
collect 0
load 1
use 1
```

This should change the settings to look for the `refcount.txt` file and load addresses from it.

Note:

- if no file `profile.txt` is found, all three options (collection, loading, using) are turned off. So it behaves as if there was no profile-based cache skipping is on.
- Please make sure the file is named `profile.txt`
- Don't forget to change the options after doing one run (i.e. data collection, and then the second run is load+use)
- The order of the options should not matter
- Only use the examples shown above. Other combinations are not implemented.
- The format is `<option> <1/0>` for turning that option on or off.

Implementation Details

The majority of the code involves utilities - reading from the file, storing into a map, loading from the map, etc. Following is a broad overview of how things are structured.

Every shader_core_ctx (SIMT Core) contains a map. The key for the map is the address reference and the kernel ID, and the value is the number of times it was referenced:

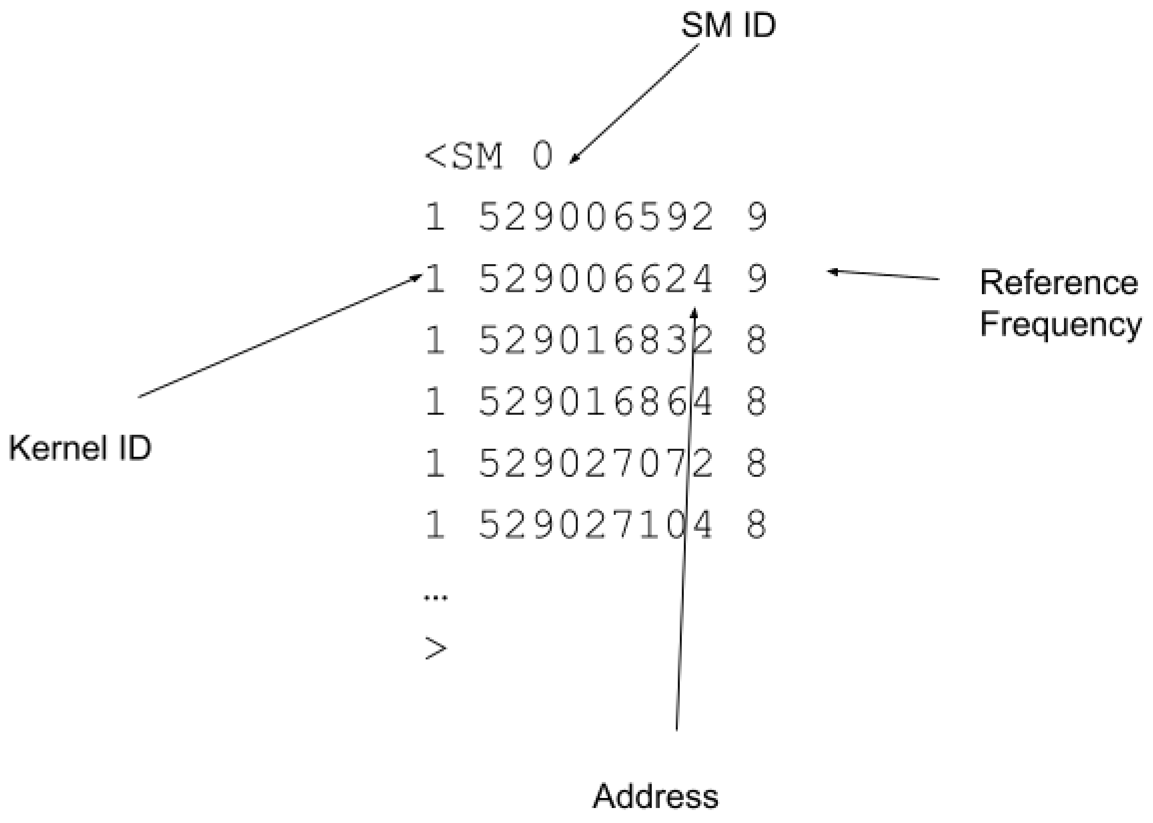
```
<address, kernel_id>  = count
```

Since this implementation focuses very precisely on the address and the kernel ID, we lose improvements from cache locality as a result of different kernels being brought into the cache.

When we are collecting those references, that data structure is updated every time a memory reference is made in the load_store unit. Similarly, when we need to load the profiled data and reference the frequency of each address, we open the `refcount.txt` file and load each SM separately and fill up the aforementioned data structure.

When we need to bypass cache using the count, we check the data structure for the number of times the address was used. Anything less than 3 is bypassed.

The data collection file, `refocunt.txt` contains information in the following format. This file is autogenerated whenever the `collect` option is turned on the `profile.txt` file.

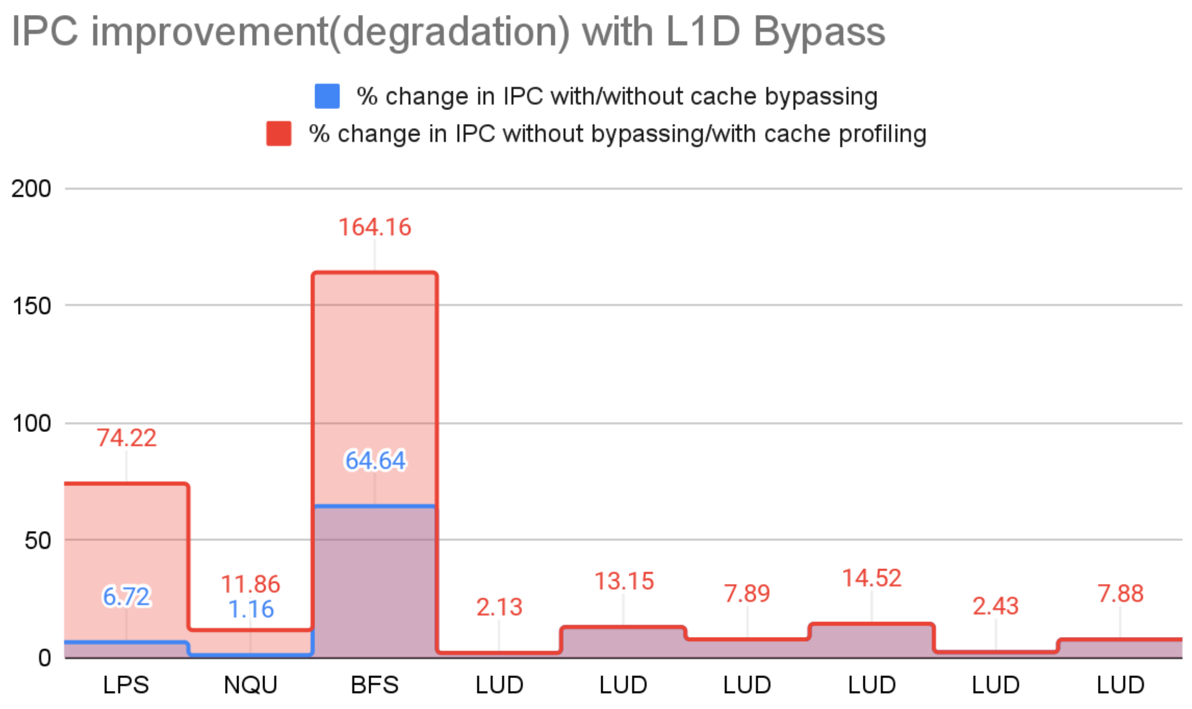


```
<SM 0
1 529006592 9
1 529006624 9
1 529016832 8
1 529016864 8
1 529027072 8
1 529027104 8
...
>
```

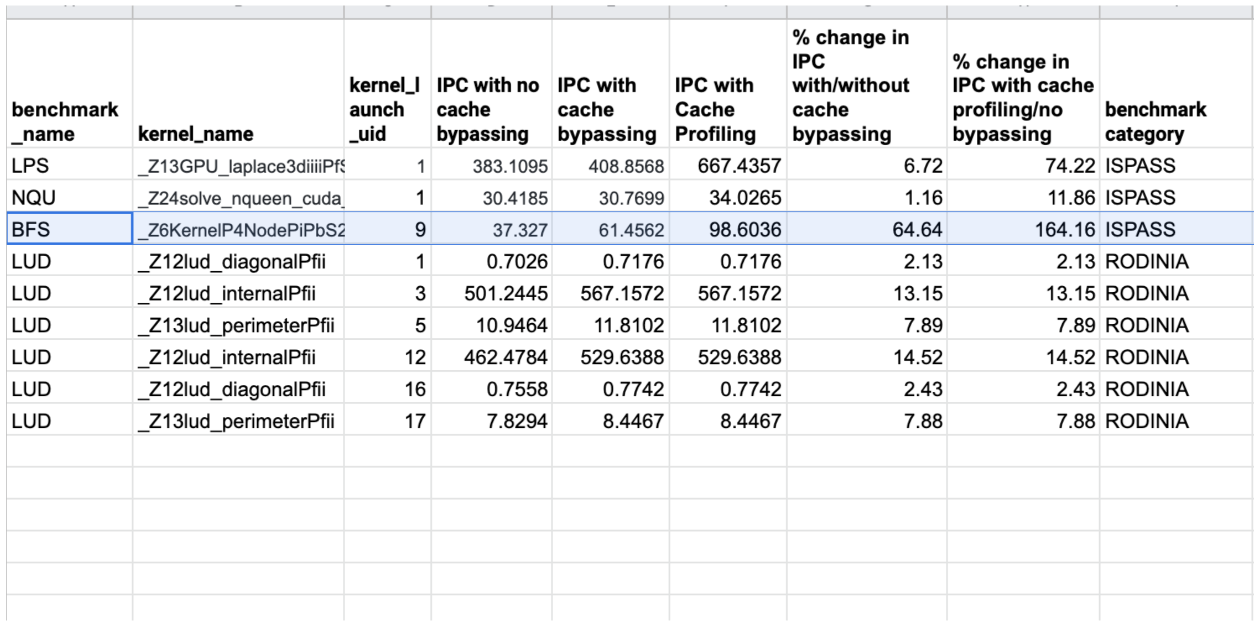
Data Collection & Analysis

Following is a graph after running profile-based cache bypassing. The blue bars indicate speedup due to cache bypassing turned on (from task1). The Red bar indicates improvement with profile-based bypassing enabled.

LPS, NQU, and BFS has seen major improvements. LUD did not change at all - perhaps this was a cache-insensitive benchmark.



And the following are the data points for the above graph



benchmark _name	kernel_name	kernel_launch_uid	IPC with no cache bypassing	IPC with cache bypassing	IPC with Cache Profiling	% change in IPC with/without cache bypassing	% change in IPC with cache profiling/no bypassing	benchmark category
LPS	_Z13GPU_laplace3diiiiPfS	1	383.1095	408.8568	667.4357	6.72	74.22	ISPASS
NQU	_Z24solve_nqueen_cuda_	1	30.4185	30.7699	34.0265	1.16	11.86	ISPASS
BFS	_Z6KernelP4NodePiPbS2	9	37.327	61.4562	98.6036	64.64	164.16	ISPASS
LUD	_Z12lud_diagonalPfii	1	0.7026	0.7176	0.7176	2.13	2.13	RODINIA
LUD	_Z12lud_internalPfii	3	501.2445	567.1572	567.1572	13.15	13.15	RODINIA
LUD	_Z13lud_perimeterPfii	5	10.9464	11.8102	11.8102	7.89	7.89	RODINIA
LUD	_Z12lud_internalPfii	12	462.4784	529.6388	529.6388	14.52	14.52	RODINIA
LUD	_Z12lud_diagonalPfii	16	0.7558	0.7742	0.7742	2.43	2.43	RODINIA
LUD	_Z13lud_perimeterPfii	17	7.8294	8.4467	8.4467	7.88	7.88	RODINIA

Appendix

In the submission zip, for task1 folder, the files follow `` `<benchmark_name><suite><l1d_skipped or not> `` format.

For Task2, you will see 3 different types of files:

1. `<becnhamrk>_collect.txt`: This file contains the result of the run when we are only collecting address reference data
2. `<benchmark>_use.txt:` This file contains the result from the run that uses the loaded data
3. `<benchmark>_refcount.txt`: The intermediate file where the reference counts are stored.