ECE 786 : Final Project
Cache Bypass Study

1. Task 1: Rodinia and ISPASS Benchmarks Analysis

benchmark name	kernel _name	kernel_launch _uid	IPC with no cache bypassing	IPC with cache bypassing	Percentage change of comparing the IPC with/without cache bypassing	benchmark category
Rodinia	BP	1	670.1913	666.3648	-0.57%	cache insensitive
Rodinia	BP	2	424.7120	192.2678	-54.72%	cache friendly
Rodinia	HS	1	701.3718	707.6299	0.89%	cache insensitive
Rodinia	LUD	1	0.7026	0.7176	2.13%	cache insensitive
Rodinia	LUD	2	9.2446	9.1103	-1.45%	cache insensitive
Rodinia	LUD	3	501.2445	567.1572	13.14%	cache unfriendly
Rodinia	LUD	4	0.7558	0.7742	2.43%	cache insensitive
Rodinia	LUD	5	10.9464	11.8102	7.89%	cache unfriendly
Rodinia	LUD	6	497.3745	574.7466	15.55%	cache unfriendly
Rodinia	LUD	7	0.7558	0.7741	2.42%	cache insensitive
Rodinia	LUD	8	10.1697	10.9718	7.88%	cache unfriendly
Rodinia	LUD	9	473.0808	557.2787	17.79%	cache unfriendly
Rodinia	LUD	10	0.7558	0.7741	2.42%	cache insensitive
ISPASS	BFS	1	217.5687	167.9066	-22.82%	cache friendly
ISPASS	BFS	2	206.0139	146.9099	-28.68%	cache friendly
ISPASS	BFS	3	165.9271	112.0179	-32.48%	cache friendly
ISPASS	BFS	4	76.2236	61.3361	-19.53%	cache friendly

ISPASS	BFS	5	21.3021	36.1667	69.77%	cache unfriendly
ISPASS	BFS	6	22.5533	44.4395	97.04%	cache unfriendly
ISPASS	BFS	7	46.5675	86.5094	85.77%	cache unfriendly
ISPASS	BFS	8	354.4445	455.3303	28.46%	cache unfriendly
ISPASS	BFS	9	473.1056	486.7920	2.89%	cache insensitive
ISPASS	NQU	1	30.4185	30.7699	1.15%	cache insensitive
ISPASS	LPS	1	383.1095	408.8568	6.72%	cache unfriendly

NOTE: Assuming less than **5%** difference is cache insensitive

Cache Unfriendly Benchmarks: **BFS** (uid: 5,6,7,8) and **LPS**

2. Task 2: Profile based bypassing

The profile based bypassing is done in two steps. In the first run of the simulation I'm collecting the per SM profile in the following format

< SM 0, kernel 1 : addr1 ref, addr2 ref, ...addrn ref
kernel 2 : addr1 ref, addr2 ref, ...addrn ref
...
kernel n : addr1 ref, addr2 ref, ...addrn ref >
...
< SM n, kernel 1 : addr1 ref, addr2 ref, ...addrn ref
kernel 2 : addr1 ref, addr2 ref, ...addrn ref
...
kernel n : addr1 ref, addr2 ref, ...addrn ref >

I'm dumping these stats in a file ***"profile.dump"*** for cache unfriendly benchmarks. For this task I'm using **BFS** and **LPS** as they showed significant change in IPC when bypassing L1 D cache.

2.1 Dumping Profiling Statistics

To dump the profiling statistics, I used a hashmap to store the key-value pairs. Then in ***shader.cc*** I'm implementing the profiling logic to collect the statistics.

```
#define READ_PACKET_SIZE 8

// WRITE_PACKET_SIZE: bytes: 6 address, 2 miscelaneous.
#define WRITE_PACKET_SIZE 8

#define WRITE_MASK_SIZE 8

extern std::map<int, std::map<int, std::map<unsigned, int>>> SM_profile_stats_map; // hash map for collecting profile stats
extern std::map<int, std::map<int, std::map<unsigned, int>>> profile_bypass_map; // hash map for profiled bypass

class gpgpu_context;

enum exec_unit_type_t {
  NONE = 0,
```

Fig 1: declaring the hash map in shader.h in gpgpu-sim

```
mem_stage_stall_type stall_cond = NO_RC_FAIL;
const mem_access_t &access = inst.accessq_back();

SM_profile_stats_map[m_core->get_sid()][m_core->get_kernel()->get_uid()][access.get_addr()] += 1;
```

Fig 2: Collecting profiling stats in shader.cc in gpgpu-sim

Using C++ I/O file support, I'm dumping the profile stats in the "***profile.dump***" file for both LPS and BFS benchmarks.

```
std::ofstream profile_stat;
profile_stat.open("profile.dump");
for(auto SM_it = SM_profile_stats_map.begin(); SM_it != SM_profile_stats_map.end(); SM_it++){
  printf("\n< SM %d :", SM_it->first);
  profile_stat << "SM " << SM_it->first << " : " << std::endl;
  for(auto kernel_it = SM_it->second.begin(); kernel_it != SM_it->second.end(); kernel_it++){
    printf("\nkernel %d : ", kernel_it->first);
    profile_stat << "kernel " << kernel_it->first << " : " << std::endl;
    for(auto addr_it = kernel_it->second.begin(); addr_it != kernel_it->second.end(); addr_it++){
      printf("%u %d, ", addr_it->first, addr_it->second);
      profile_stat << "addr : " <<addr_it->first << ", ref : " << addr_it->second << ", " << std::endl;
    }
  }
  profile_stat << std::endl;
  printf(">\n");
}
```

Fig 3: Dumping the profile stats in profile.dump

Command:
./ispass-2009-BFS graph65536.txt (BFS)
./ispass-2009-LPS (LPS)

2.2 Implementing profile based cache bypass logic

After dumping the statistics, I'm Implementing a profile based cache bypass in shader.cc where I will bypass the L1 D cache for all those instructions with a reference counter less than 3. For this I will be using **SM7_QV100** GPU configuration.

```
std::ifstream profile_stat;
profile_stat.open("profile.dump");
if (profile_stat.is_open())
{
  std::cout << "Reading the profile stats" << std::endl;
  std::string line;
  std::string data_type;
  int SM_id = -1, kernel_id = -1, addr = -1, ref = -1; // Initialize to some default values
  while (getline(profile_stat, line))
  {
    if (line.empty())
      continue;
    std::istringstream iss(line);
    iss >> data_type;
    if (data_type == "SM")
    {
      iss.ignore(std::numeric_limits<std::streamsize>::max(), ' '); // Ignore until the first space
      iss >> SM_id;
    }
    else if (data_type == "kernel")
    {
      iss.ignore(std::numeric_limits<std::streamsize>::max(), ' '); // Ignore until the first space
      iss >> kernel_id;
    }
    else if (data_type == "addr")
    {
      std::string temp;
      iss >> temp >> addr >> temp >> ref; // Read addr and ref directly from stream
      profile_bypass_map[SM_id][kernel_id][addr] = ref;
    }
  }
  profile_stat.close();
}
```

Fig 4: Reading the reference counter in gpgpusim-entrypoint.cc

```
if (profile_bypass_map[m_core->get_sid()][m_core->get_kernel()->get_uid()][access.get_addr()] < 3){
  inst.cache_op = CACHE_GLOBAL;
}
```

Fig 5: profile based cache bypassing logic

2.3 Results

benchmark	kernel_uid	IPC before profiling	IPC with profile based cache bypass	% IPC change
BFS	5	87.2393	66.7220	-26.65%
BFS	6	86.6631	66.8945	-25.74%
BFS	7	145.4857	111.0300	-26.86%
BFS	8	229.1067	175.9334	-26.25%
LPS	1	638.116	667.4357	4.49%

We can see that the profile based cache bypassing has less IPC than before profiling. So for our unfriendly cache benchmark, BFS does not improve from having a profile based cache bypass. LPS has a considerable effect of profile bypass.

NOTE: This simulation was carried in SM_QV100 GPU configuration.