ECE 786 Final project report
Chandu Yuvarajappa
cyuvara

TASK 1

The following are the benchmarks I have categorized:
If the range of the IPC change from cache bypass to no cache bypass is between -5 and +5%,CACHING INSENSTIVE
If the difference between IPC with no cache bypass and cache bypass is greater than 5%, CACHE UNFRIENDLY
If there is a 5% decrease in IPC from cache bypass to no cache bypass, CACHE FRIENDLY

RESULTS FOR ISPASS BENCHMARKS:
From ISPASS, I have chose benchmarks BFS,LPS,NQU.

BFS

Benchmark name	Kernel_name	Kernel_launch_uid	IPC without cache bypassing	IPC with cache bypassing	Change in IPC with/without cache bypassing in percentage	category
BFS	_Z6KernelP4NodePiPbS2_S1_S2_i	1	217.5687	167.9066	-22.82	FRIENDLY
	_Z6KernelP4NodePiPbS2_S1_S2_i	2	206.0139	146.9099	-28.69	FRIENDLY
	_Z6KernelP4NodePiPbS2_S1_S2_i	3	165.9271	112.0179	-32.49	FRIENDLY
	_Z6KernelP4NodePiPbS2_S1_S2_i	4	76.2236	61.3361	-19.5	FRIENDLY
	_Z6KernelP4NodePiPbS2_S1_S2_i	5	21.3021	36.1667	69.78	UNFRIENDLY
	_Z6KernelP4NodePiPbS2_S1_S2_i	6	22.5533	44.4395	97.04	UNFRIENDLY
	_Z6KernelP4NodePiPbS2_S1_S2_i	7	46.5675	86.5094	85.77	UNFRIENDLY
	_Z6KernelP4NodePiPbS2_S1_S2_i	8	354.4445	455.3303	28.46	UNFRIENDLY
	_Z6KernelP4NodePiPbS2_S1_S2_i	9	473.1056	486.792	2.89	INSENSTIVE
Total			37.327	61.4562	64.64	UNFRIENDLY

LPS

Benchmark name	Kernel_name	Kernel_launch_uid	IPC without cache bypassing	IPC with cache bypassing	Change in IPC with/without cache bypassing in percentage	category

LPS	__Z13GPU_laplace3diiiiPfS_	1	383.1095	408.8568	6.72	UNFRIENDLY

NQU

Benchmark name	Kernel_name	Kernel_launch_uid	IPC without cache bypassing	IPC with cache bypassing	Change in IPC with/without cache bypassing in percentage	category
NQU	___Z24solve_nqueen_cuda_kerneliiPjS_S_S_i	1	30.4185	30.7699	1.16	UNFRIENDLY

In ISPASS, only BFS & LPS are proved to be cache unfriendly

RESULTS FOR RODINIA BENCHMARKS:
BP:

Benchmark name	Kernel_name	Kernel_launch_uid	IPC without cache bypassing	IPC with cache bypassing	Change in IPC with/without cache bypassing in percentage	category
BP	_Z22bpnn_layerforward_CUDAPfS_S_S_ii	1	675.6067	671.3728	-0.63	INSENSTIVE

HS:

Benchmark name	Kernel_name	Kernel_launch_uid	IPC without cache bypassing	IPC with cache bypassing	Change in IPC with/without cache bypassing in percentage	category
HS	_Z14calculate_tempiPfS_S_iiiiffffff	1	701.3718	707.6299	0.89	INSENSTIVE

LUD
Statistics for each kernel.
KERNEL NAME: _Z12lud_diagonalPfii

Benchmark name	Kernel name	Kernel launch uid	Ipc without cache bypassing	Ipc with cache bypassing	Change in IPC with/without cache bypassing in percentage	category
LUD	_Z12lud_diagonalPfii	1	0.7026	0.7176	2.13	INSENSTIVE
		4	0.7558	0.7742	2.43	INSENSTIVE
		7	0.7558	0.7741	2.42	INSENSTIVE
		10	0.7558	0.7741	2.42	INSENSTIVE
		13	0.7558	0.7741	2.42	INSENSTIVE
		16	0.7558	0.7741	2.42	INSENSTIVE
		19	0.7558	0.7741	2.42	INSENSTIVE
		22	0.7558	0.7741	2.42	INSENSTIVE
		25	0.7558	0.7741	2.42	INSENSTIVE
		28	0.7558	0.7741	2.42	INSENSTIVE
		31	0.7558	0.7741	2.42	INSENSTIVE
		34	0.7558	0.7741	2.42	INSENSTIVE
		37	0.7558	0.7741	2.42	INSENSTIVE
		40	0.7558	0.7741	2.42	INSENSTIVE
		43	0.7558	0.7741	2.42	INSENSTIVE
		46	0.7558	0.7741	2.42	INSENSTIVE
TOTAL	_Z12lud_diagonalPfii					INSENSTIVE

Overall each kernel for this kernel type is cache insensitive. So, Overall the kernel _”Z12lud_diagonalPfii” is **CACHE_INSENSITIVE.**

KERNEL NAME: _Z12lud_internalPfii

Benchmark name	Kernel name	Kernel launch uid	Ipc without cache bypassing	Ipc with cache bypassing	Change in IPC with/without cache bypassing in percentage	category
LUD	_Z12lud_internalPfii	3	501.2445	567.1572	13.15	UNFRIENDLY
		6	497.3745	574.7466	15.55	UNFRIENDLY
		9	473.0808	557.2787	17.97	UNFRIENDLY
		12	462.4784	529.6388	14.52	UNFRIENDLY
		15	378.4012	504.6895	33.37	UNFRIENDLY
		18	357.2093	493.737	38.22	UNFRIENDLY
		21	338.0277	453.3258	34.12	UNFRIENDLY
		24	324.1251	467.1097	44.11	UNFRIENDLY
		27	290.9933	405.207	39.25	UNFRIENDLY
		30	246.8571	344.3503	39.50	UNFRIENDLY

		33	208.6225	252.2766	21.40	UNFRIENDLY
		36	142.2966	172.1319	20.97	UNFRIENDLY
		39	111.9498	134.8471	20.45	UNFRIENDLY
		42	39.4499	44.3208	13.87	UNFRIENDLY
		45	16.2623	16.6957	2.66	INSENSTIVE
TOTAL	_Z12lud_internalPfii					UNFRIENDLY

All kernels except kernel 45 launched for this kernel type are CACHE UNFRIENDLY. So, Overall the kernel "_Z12lud_internalPfii" is **CACHE_UNFRIENDLY.**

KERNEL NAME: _Z13lud_perimeterPfii

Benchmark name	Kernel name	Kernel launch uid	Ipc without cache bypassing	Ipc with cache bypassing	Change in IPC with/without cache bypassing in percentage	category
LUD	_Z13lud_perimeterPfii	2	9.2446	9.1103	-1.45	INSENSTIVE
		5	10.9464	11.8102	7.89	UNFRIENDLY
		8	10.1697	10.9718	7.89	UNFRIENDLY
		11	9.3893	10.1287	7.89	UNFRIENDLY
		14	8.6082	9.2874	7.89	UNFRIENDLY
		17	7.8294	8.4467	7.89	UNFRIENDLY
		20	7.0473	7.604	7.89	UNFRIENDLY
		23	6.264	6.7609	7.93	UNFRIENDLY
		26	5.4832	5.9163	7.89	UNFRIENDLY
		29	4.7006	5.0733	7.92	UNFRIENDLY
		32	3.9172	4.2288	7.95	UNFRIENDLY
		35	3.1348	3.3833	7.92	UNFRIENDLY
		38	2.3514	2.5387	7.96	UNFRIENDLY
		41	1.5679	1.6926	7.95	UNFRIENDLY
		44	0.8583	0.8467	-1.95	INSENSTIVE
TOTAL	_Z13lud_perimeterPfii					UNFRIENDLY

All kernels except for kernels 2,44 launched for this kernel type are CACHE UNFRIENDLY. So, Overall the kernel " _Z13lud_perimeterPfii" is **CACHE_UNFRIENDLY.**
In RODINIA only LUD is cache unfriendly.
Out of six benchmarks I have tested, Only BFS,LPS,LUD are CACHE UNFRIENDLY.

TASK 2:

FIRST RUN OF SIMULATION

I updated the ldst_unit::memory_cycle() method in shader.cc as follows to profile the number of accesses for each address for each kernel processed by shader.
Shader_id, Kernel_id, Address, and Reference_Count are the indexes for my multi-dimensional associative map.The number of accesses to each location is represented by the reference count.

```
typedef unsigned long long addr_type;
typedef std::map<addr_type,int> addr_to_ref_cnt_map;
typedef std::map<int,addr_to_ref_cnt_map> kernel_to_addr_map;
typedef std::map<int,kernel_to_addr_map> shaderid_to_kernel_map;
shaderid_to_kernel_map sid_kernel_addr_ref_mapping;
```

In the shader_cluster (SIMT_Cluster), each LDST unit is connected to a specific shader_core (SIMT core). By default, LDST has a pointer to the corresponding shader_core_ctx (m_core). We can access the kernel that the core is presently using using the get_kernel() function, and we can acquire the kernel_id using the get_id() function in the kernel.

```
bool ldst_unit::memory_cycle(warp_inst_t &inst,
                             mem_stage_stall_type &stall_reason,
                             mem_stage_access_type &access_type) {
  kernel_info_t *kernel ;
  kernel = m_core->get_kernel();
  int kernel_id = kernel->get_uid();
```

Obtaining access information in memory_cycle():

```
  if(sid_kernel_addr_ref_mapping[m_sid]
[kernel_id].find(access.get_addr())==sid_kernel_addr_ref_mapping[m_sid][kernel_id].end())
    sid_kernel_addr_ref_mapping[m_sid][kernel_id][access.get_addr()]=1;
  else
    sid_kernel_addr_ref_mapping[m_sid][kernel_id][access.get_addr()]++;
```

I have printed the profiling data into the file "results_counter.txt" in the gpgpu_sim::shader_print_cache_stats function.FOUT (FILE)

```
    //added for final project
    for(auto & sid_map_pair : sid_kernel_addr_ref_mapping) {
      for(auto & kernel_map_pair : sid_map_pair.second) {
         for(auto & addr_map_pair : kernel_map_pair.second){
              fprintf(fp,"%d %d %llx %d
\n",sid_map_pair.first,kernel_map_pair.first,addr_map_pair.first,addr_map_pair.second);
         }
      }
    }
    fclose(fp) ;
    //added for final project
```

Format of results_counter.txt generated:

```
shader_id  kernel_id   address       number of access for address
0           1          c0000000        18
```

SECOND RUN OF SIMULATION

The following files are changed for the second simulation run:

shader.cc
gpu-sim.cc
gpu-sim.h

I developed a similiar multi-dimensional associative map sid_kernel_addr_ref_mapping of the same type shaderid_to_kernel_map (as used in the first phase) in the gpgpu_sim class to load the local reference statistics in the simulator. I parsed the results_counter.txt file created in the first simulation phase during the class instantiation (I had already moved the results_counter.txt file received in the first simulation phase of task2 into the run folders of the second simulation phase).

In the gpgpu_sim constructor, the following code may be found: gpgpu_sim::gpgpu_sim(const gpgpu_sim_config &config, gpgpu_context *ctx).

```
//added for final project
  FILE *fp ;
  int shader_id;
  int kernel_id;
  addr_type addr;
  int ref_cnt;
  fp = fopen("results_counter.txt", "r") ;
  while(fscanf(fp, "%d %d %llx %d", &shader_id, &kernel_id,&addr,&ref_cnt) != EOF){
      sid_kernel_addr_ref_mapping[shader_id][kernel_id][addr]=ref_cnt;
  }
```

Changes in shader.cc**, ldst_unit::memory_cycle():**

```
                                    mem_stage_
//added for the final project
  gpgpu_sim *gpu_pointer;
  gpu_pointer = m_core->get_gpu();
  kernel_info_t *kernel ;
  kernel = m_core->get_kernel();
  int kernel_id = kernel->get_uid();

//added for the final project
```

The pointer(m_core) to the instance of the shader_core_ctx class it is connected to is contained in the ldst_unit. To obtain the matching gpgpu_sim class pointer, use the get_gpu() function in the m_core module. The get_kernel() function can be used to acquire kernel class information. The kernel class pointer's get_uid() function is used to obtain the kernel_id. By default, the class member of ldst_unit with the name m_sid is shader_id.

```
//added for final project
  if(gpu_pointer->sid_kernel_addr_ref_mapping[m_sid][kernel_id][ access.get_addr()]<3)bypassL1D=true;
//added for final project
```

When the local reference counter value is 3, a choice to bypass is made for each address matching to its shader_id and kernel_id.

ANALYSIS

The results of the two simulations for the cache-unfriendly configurations BFS,LPS,LUD are shown in the table below.

To profile data during the first simulation run (no bypass is used).

After loading profiled data, the second simulation run (bypass is applied is cnt 3)

Both simulations are carried out with the setting -gpgpu_gmem_skip_L1D=0.

	1st run	2nd run
BFS	gpu_tot_ipc	gpu_tot_ipc
_Z6KernelP4NodePiPbS2_S1_S2_i	129.8383	128.0733
LPS		
_Z13GPU_laplace3diiiiPfS_	638.116	667.4357
LUD		
_Z12lud_diagonalPfii	0.7678	0.7782
_Z12lud_internalPfii	291.96	295.34
_Z13lud_perimeterPfii	7.818	8.239
Overall for LUD	38.7866	39.8571

BFS: An unexpected little decline in IPC from 129.833 to 128.0733 was observed. I believe this is a result of the cache-friendly kernels contained in BFS (we can see that kernels 1, 2, and 3 for BFS are cache-friendly from job 1). Cache-friendly kernels outperform cache-unfriendly kernels in terms of performance while skipping cache data.

IPC has clearly increased from 638.116 to 667.4357, according to LPS.

LUD: All kernel types have improved IPC in LUD. IPC therefore rises from 38.7866 to 39.8571.

For each type of kernel, gpu_sim_insn / gpu_sim_cycle is used to determine the gpu_tot_ipc for each individual kernel.

SUBMISSION FORMAT:

task1 directory:

sub-directories:

❼withoutbypass (contains result files for each benchmark saved by corresponding bench mark name)

→BFS
→LPS
→NQU
→BP
→HS
→LUD

❼withbypass (contains result files for each benchmark marked by corresponding bench mark name)

→BFS
→LPS
→NQU
→BP
→HS
→LUD

TASK 2 Directory:

Results of profiling are contained in the set1 directory.

1. Has BFS, LPS, and LUD directories. Both result.txt (simulation results acquired without bypass) , results_counter.txt (profiled data generated in step1) and results_bfs_graph65536(output file) are contained in each folder.

Shader.cc 2. (This file has been changed to allow for local SM profiling.)

set1 directory: (Profiled data loaded into set2 directory; results)

1. Has folders for BFS, LPS, and LUD.Results_counter.txt (profiled data file copied from set1) and result.txt (simulation results after local bypass) are both contained in each folder.

gpu-sim.cc, gpu-sim.h, and shader.cc According to the statistics gathered, these files have been updated to load results_counter.txt and allow local bypass.

Please transfer the "results_countert.txt" file generated in step 1 to the appropriate run directories in step 2. Because this file is automatically read in step 2, "make" fails if it is not present in the set directory.