EECE 5640 Final Project - Report

by Keelin Becker-Wheeler

April 17, 2019

In this project, I use gpgpu-sim [4] to simulate various set indexing methods for GPU caches. I performed simulations of four set indexing functions on four varying models of a GPU cache, giving 16 different total system configurations. I analyzed the performances of these configurations using 12 separate benchmarks.

I have uploaded the code for this project on github:
https://github.com/keelimeguy/eece5640-final

Contents

1 Environment

I used my personal computer to run all simulations, see My System Info. The approximate steps one can use to setup the environment for this project are described in the Makefile. That makefile script describes some (not necessarily all) of the dependencies needed (notably gcc version 4.4 and makedepend), installation of the necessary CUDA tools[3] (cudatoolkit_4.0.17 and gpucomputingsdk_4.0.17), setup of the gpgpu-sim and benchmark source repositories (gpgpu-sim_distribution[5] and ispass2009-benchmarks[6]) along with the source code changes I made to allow compilation of those repositories on my machine.

2 Configurations

I used 16 total system configurations during my simulations. These configurations are setup by editing the gpgpusim.config file used by gpgpu-sim, based on the GTX480 configuration files given in the source code repository[5]. The configuration options which are varied are those of the L1 data cache, given by the "-gpgpu_cache:dl1" line in the config file. The following is the syntax used for this configuration:

-gpgpu_cache:dl1
    nset : line_sz : assoc,
    repl_policy : write_policy : alloc_policy : write_alloc_policy : set_idx_func,
    mshr_type : mshr_entries : mshr_max_merge,
    miss_queue_size : result_fifo_entries,
    data_port_width

Here as an example usage, where the possible option codes (the letters) and their meanings can be found by studying the gpu-cache.h file that comes with the gpgpu-sim source code[5] (I define the relevant options to this project further below):

-gpgpu_cache:dl1 64:128:6,F:L:m:N:L,A:32:8,8

The parameters I vary are the number of sets, the replacement policy, and the set index function. I vary the number of sets to be 32 or 64, and used either a FIFO (F) or LRU (L) replacement policy. Additionally, a 16kB cache with 4-way associativity is used when there are 32 sets, while a 48kB cache with 6-way associativity is used when there are 64 sets. This variance in the cache size is meant to mimic the available configurations of the Fermi GPU architecture, in which 64kB of memory can be split among the L1 cache and shared memory. This is changed using the "-gpgpu_shmem_size" line option of the config file. The process of varying the configuration options is shown in the Run Script, used to run the simulations. The set index functions used are described in the next section.

3 Set Index Functions

Before describing the indexing functions, it is important to review set indexing. Set indexing is basically the mapping of a large space of memory addresses to a smaller space of cache locations. Similar to the concept of hashing in computer science, this mapping should be uniformly distributed in order to reduce address collisions. Collisions in cached memory result in older addressed values being replaced with new addressed values. If the older addresses are then needed after they were replaced, then they would be fetched from the main memory in a much slower process than if they were still present in the cache. Thus, reducing the number of collisions is the goal.

For every mapped cache location (called a cache set), a number of addresses are stored based on the associativity of the cache. After a number of collisions fill every available space given by this associativity, a replacement algorithm is used to determine which of the stored values will be replaced on further collisions of the cache set. When designing the function which maps addresses to these cache sets, it would be ideal to

know the access order of addresses in order to minimize the number of collisions that will occur. In reality, this function is only implemented once without knowledge of every program that may run on the system, and so must work efficiently for as many orderings of address accesses as possible.

For the functions described below, the following address components are assumed:

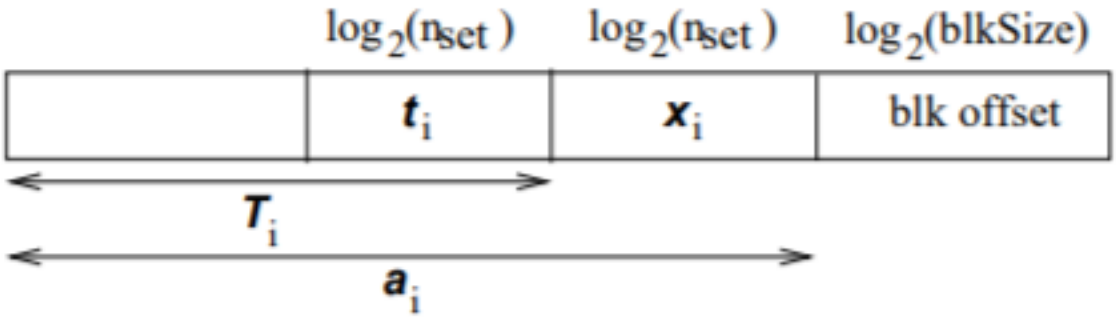


image source: Using Prime Numbers for Cache Indexing to Eliminate Conflict Misses[8]

That is, the address is split, starting from the right, into a number of bits for the block offset (determined by line size), a number of bits for the cache index (determined by number of sets), and the remainder of the address (called the tag) which, in this case, is larger in size than the cache index.

3.1 Linear Set Function (L)

The linear set indexing function gives one of the simplest mappings of addresses to cache sets. Simply, the first address maps to the first set, then the next address to the second set, and so on until all sets have been covered. After that point, the next address will then map again to the first set and the process repeats. This process is described by the simple algorithm: $set\_idx = a \ \% \ nset = x$

3.2 Simple XOR Set Function (S)

The XOR set indexing function attempts to map the addresses to cache sets in a less predictable pattern, by incorporating some of the tag bits in the algorithm that determines the set index. Simply, the cache index bits of the address are XOR-ed with an equal number of tag bits: $set\_idx = x \oplus t$

3.3 Fermi Hash Set Function (F)

This set indexing function comes from the paper: A Detailed GPU Cache Model Based on Reuse Distance Theory[2]. In that paper, the authors run several micro-benchmarks on a Fermi-architecture NVIDIA GPU with the goal of reproducing a model of the GPU cache. After analysis of the results of their benchmarks, the authors construct a model of the Fermi GPU cache, including a set indexing function. The following figure (taken from their paper) describes the implementation of the function:

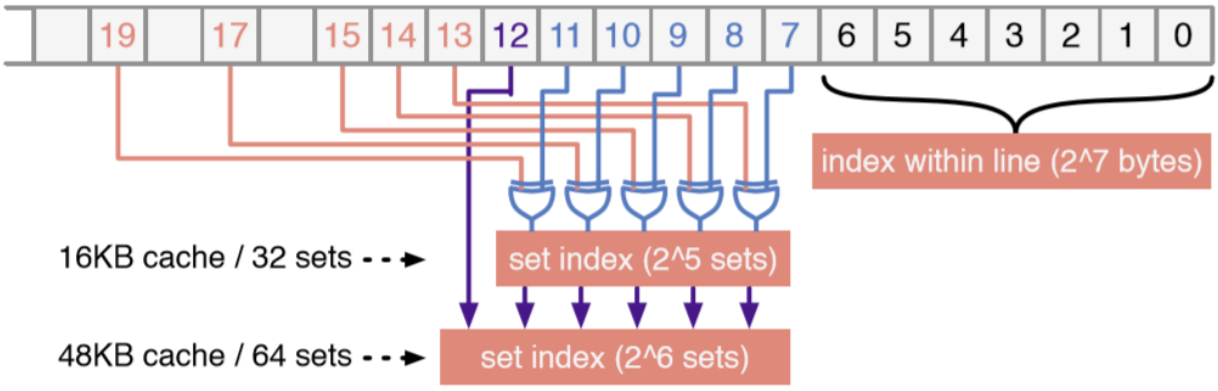


image source: A Detailed GPU Cache Model Based on Reuse Distance Theory[2]

We see that the set indexing function derived from the Fermi architecture is very similar to the Simple XOR function. The main difference here is that the used tag bits are not all adjacent. Differences are also seen in the hash function depending on whether a 32-set or 64-set configuration is used. In the 64-set configuration, the extra set index bit of the address is simply appended to the final set index.

3.4 Psuedo-Random Interleaved Set Function (P)

The intuition for this set indexing function begins by referencing hashing in computer science: when hashing using a modulus operator, it is best to use a prime number as a base. Doing otherwise can result in the placement of all numbers of a certain multiple into the same position. This would be bad in terms of caching as an address access pattern with a stride equal to that multiple could map every value to a single cache set. In order to more uniformly distribute the mapping of addresses, a psudeo-random function should be derived using the notion of prime numbers. [8]

The paper Pseudo-randomly Interleaved Memory[1] by B. R. Rau describes an algorithmic process that can be used to derive such a function for use in caches. The algorithms described make heavy use of polynomial math, including modulus operations between polynomial functions. In this type of math, irreducible polynomials become synonymous to prime numbers. Additionally, in order to use the algorithms in the context of caches, only two numbers may be used (1 or 0) and all other numbers resulting from arithmetic need to be reduced to these numbers (odd numbers become 1, even become 0). The use of only these two numbers will be denoted by GF(2) — if you've studied group theory, GF(2) is equivalent to the group consisting of 0 and 1 where the addition operator is mapped to XOR and the multiplication operator is mapped to AND. From the paper, a derived algorithm of interest to us is:

$$B(x) = (A(x)/x^m) * x^m + (A(x) \bmod P(x))$$

where:
$A(x) = \sum_{i=0}^{n-1}(a_i x^i)$, a_i is the i-th bit of the address (of n-bits)
$B(x) = \sum_{i=0}^{m-1}(b_i x^i)$, b_i is the i-th bit of the final set index (of m-bits)
and $P(x)$ is an irreducible polynomial of degree m over GF(2).

If you're following this math well enough, then you might notice that even though this function is very useful to our problem, it would be incredibly difficult to implement in hardware. In order to reduce this indexing function into a form useable in terms of computer hardware, we define a boolean n×m matrix H, where H[i,j] = 1 iff a_{m-i-1} is an input to the XOR gate whose output is b_j. If we were able to formulate such a matrix, then we can use it to derive a simple XOR function that will compute the set index.

Rau describes in his paper[1] the derivation of such an H-matrix. The steps he describe first require a primitive element d satisfying $d^m = P(x)$ and $d^i \neq d^j$, for $0 \leq \{i, j\} < m$ and $i \neq j$ (the primitive element in this context is essentially the group theory concept of a generator). The i-th row (from the bottom) of the H-matrix is then given by d^i mod $P(x)$, where the coefficients of the resulting polynomial give the value in the associated column.

The paper Efficient Utilization of GPGPU Cache Hierarchy[7] references this algorithm to construct a set indexing function for a cache of 32-sets ($m = 5$). The authors of that paper use $P(x) = x^5 + x^2 + 1$, but failed to show their work or even note what primitive element d they used. When performing the H-matrix calculations for 32-sets myself, I used $d = x$ and happily found that my results were the same as theirs, verifying that my understanding and implementation of the processes were correct. I then performed the same H-matrix calculations for 64-sets using the parameters: $m = 6$, $d = x$, and $P(x) = x^6 + x + 1$ (including verification that $d = x$ remains a primitive element for this case). In both derivations of the set index functions, I used a maximum of 10 terms per bit function.

For my calculated H-matrices, see PRIS Derivations.
For C implementations of all the set index functions I've described here, see Set Index Function Code.

4 Benchmarks

I used many of the benchmarks found in the ispass2009-benchmarks source code repository[6]. Additionally, I wrote one other custom benchmark.

The benchmarks used were:

AES	AES cryptography benchmark
BFS	Graph traversal benchmark (Breadth First Search)
CP	Computational chemistry benchmark (Coulombic Potential)
CUSTOM1	Custom linear thrashing benchmark
LIB	Investment benchmark (LIBOR)
LPS	Laplace transform benchmark
MUM	Genome alignment benchmark (Bioinformatics)
NN	Neural Network benchmark
NQU	N-Queens chess benchmark
RAY	Ray tracing benchmark
STO	Storage systems benchmark
WP	Weather prediction benchmark

Here is the relevant code for the CUSTOM1 benchmark:

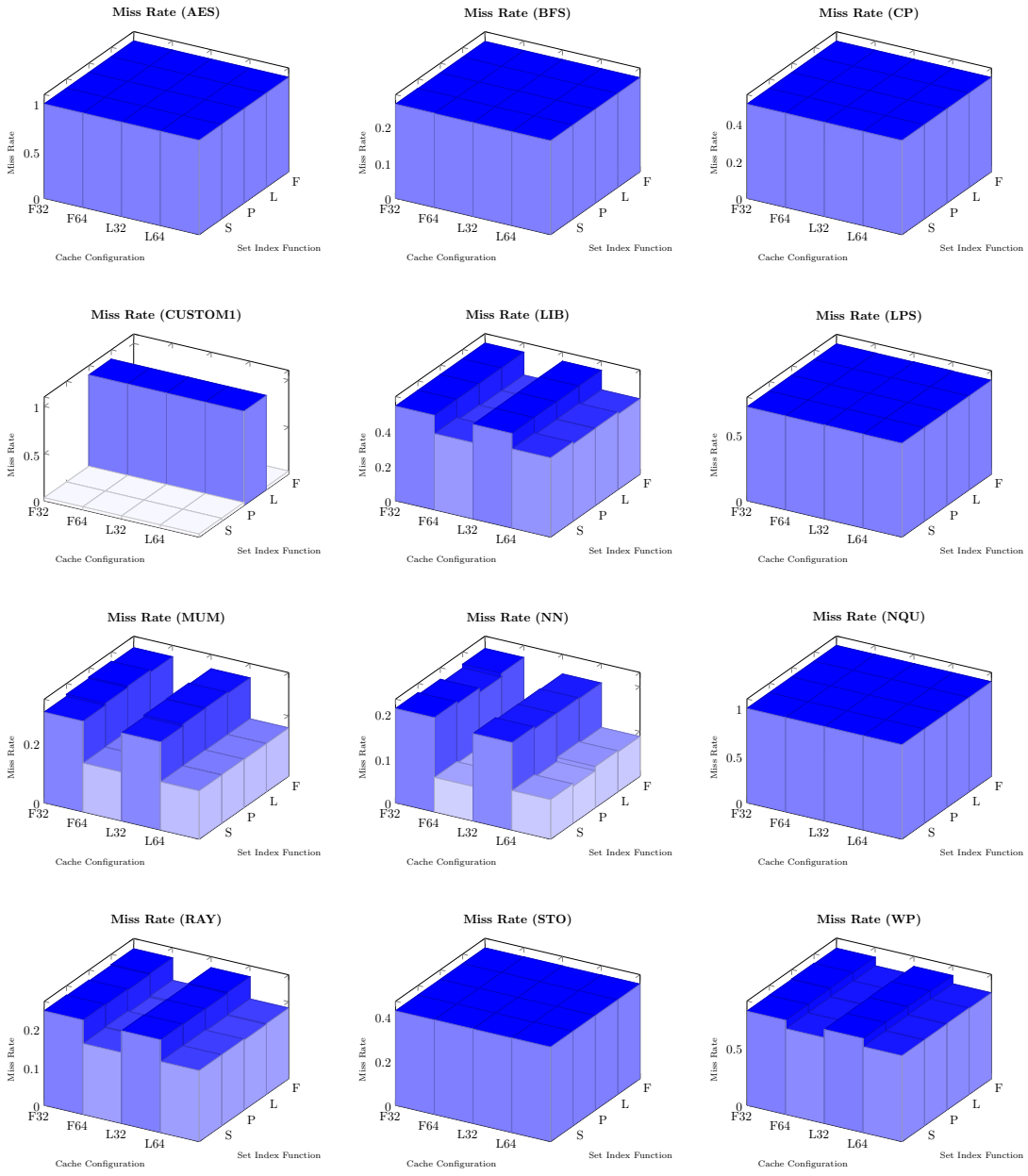
```
#define SET_SIZE 64
#define LINE_SIZE 128
#define STRIDE LINE_SIZE*SET_SIZE

__global__ void kernel(char *out, char *in, int size) {
    long tid = threadIdx.x + blockIdx.x * blockDim.x;
    if (tid < size) out[tid] = in[(tid*STRIDE)%size];
}
```

This custom benchmark simply accesses memory in a pattern which, in the case that basic linear set indexing is used, should thrash the same cache sets repeatedly.

5 Results

For each of the 12 benchmarks, I ran one simulation for each combination of cache hardware and set index function, for 16 total configurations. I recorded the cache miss rate of the L1 data cache for each simulation and generated the following graphed results:



Set Index Function: S=Simple XOR, P=Psuedo-Random Interleaved Set, L=Linear, F=Fermi Hash
Cache configuration: F=FIFO Replacement, L=LRU Replacement, 32 or 64 sets

We see that the CUSTOM1 benchmark had the intended effect of thrashing the cache for the linear set function. From the other benchmarks however, we find that changes in the set index function have minimal

effect on typical workloads. In the MUM benchmark, we see a trend where the linear indexing function performed the worst while the simple XOR function performed the best. This trend does not reappear in the other benchmarks where changes were seen, as in the RAY benchmark where it seems that the linear indexing function performed the best. Additionally, the NN benchmark was very inconsistent in which indexing function would perform the best, perhaps due to randomness in its program execution, or perhaps because the difference in choice of replacement policy had an unexpected combined effect with the set indexing function.

These results do show that the 64-set configuration performed much better for many of the benchmarks than the 32-set configuration, especially in the NN benchmark. In all other benchmarks, no differences were seen. Additionally, the choice of a FIFO or LRU replacement policy seemed to provide no differences in the miss rates of these workloads.

6 Conclusions

Based on the results, we can determine that changes in the set index function have minimal effect on the miss rate for typical workloads. It seems that the chosen workload has a large effect on the performance, and workloads can be designed to take advantage of weaknesses in any given indexing function. I would conclude that any of these indexing functions would make a good choice for a GPU cache, where a non-linear choice would likely make for a better choice.

Based on the results, we can certainly conclude that using 64 sets with 6-way associativity is a better choice than using 32 sets with a 4-way associativity. However, it is unclear whether the greater number of sets or greater associativity had the most impact on this trend. Another experiment that varied more aspects of the cache could be performed to completely determine the best cache configuration to use for these GPU workloads.

Additionally, I was surprised to find no differences in the miss rate between the use of FIFO and LRU cache replacement policies. I wonder if no differences were seen simply because the chosen workloads do not benefit from an optimization in replacement policy. If I had the time for this project, I would write and test another custom benchmark meant to highlight the difference between FIFO and LRU replacement policies. But then, I would likely only conclude again that the choice in workload has the largest effect on cache miss rate, since a workload could be designed to take advantage of any weaknesses in a given cache.

Another experiment which explicitly compared the cache behaviors of GPUs and CPUs would be interesting, but I would assume that no real differences exist between GPU and CPU caches. Perhaps the largest differences are the typical workloads that run on each type of system. In all, I think that when designing the cache for a GPU one should worry most about implementing efficient hardware. The typical workloads of GPUs so far seem to not need the most “theoretically superior” cache policies, and in cases where the choice in policy could matter, simple changes to the workload could be made instead that make the workload more compatible with the cache structure. Power efficiency and speed should be large design choices for these systems. Perhaps this is why the Fermi set indexing function turned out to be so relatively simple.

References

[1] B. R. Rau, "Pseudo-randomly interleaved memory," *Proceedings of the 18th annual international symposium on Computer architecture - ISCA 91*, 1991. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.12.7149&rep=rep1&type=pdf . [Accessed: 17-Apr-2019].

[2] C. Nugteren, G.-J. V. D. Braak, H. Corporaal, and H. Bal, "A Detailed GPU Cache Model Based on Reuse Distance Theory," *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014. [Online]. Available: https://cnugteren.github.io/downloads/Nugteren2014b.pdf . [Accessed: 17-Apr-2019].

[3] "CUDA Toolkit 4.0," *NVIDIA Developer*, 11-Oct-2018. [Online]. Available: https://developer.nvidia.com/cuda-toolkit-40 . [Accessed: 17-Apr-2019].

[4] "GPGPU-Sim," *gpgpu-sim*. [Online]. Available: http://www.gpgpu-sim.org/ . [Accessed: 17-Apr-2019].

[5] Gpgpu-Sim, "gpgpu-sim/gpgpu-sim_distribution," *GitHub*, 30-Jan-2015. [Online]. Available: https://github.com/gpgpu-sim/gpgpu-sim_distribution . [Accessed: 17-Apr-2019].

[6] Gpgpu-Sim, "gpgpu-sim/ispass2009-benchmarks," *GitHub*, 18-Feb-2013. [Online]. Available: https://github.com/gpgpu-sim/ispass2009-benchmarks . [Accessed: 17-Apr-2019].

[7] M. Khairy, M. Zahran, and A. G. Wassal, "Efficient utilization of GPGPU cache hierarchy," *Proceedings of the 8th Workshop on General Purpose Processing using GPUs - GPGPU 2015*, 2015. [Online]. Available: http://www.mzahran.com/gpgpu2015.pdf . [Accessed: 17-Apr-2019].

[8] M. Kharbutli, K. Irwin, Y. Solihin, and J. Lee, "Using Prime Numbers for Cache Indexing to Eliminate Conflict Misses," *10th International Symposium on High Performance Computer Architecture (HPCA04)*, 2004. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.409.5559&rep=rep1&type=pdf . [Accessed: 17-Apr-2019].

Appendix

My System Info

I use the Windows Subsytem for Linux, on a HP Spectre x360 - 13t-ae000 CTO laptop running Windows 10:

```
$ lsb_release -a && uname -r
No LSB modules are available.
Distributor ID: Debian
Description:    Debian GNU/Linux 9.4 (stretch)
Release:        9.4
Codename:       stretch
4.4.0-17763-Microsoft
$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):             1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 142
Model name:            Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
Stepping:              10
CPU MHz:               1992.000
CPU max MHz:           1992.0000
BogoMIPS:              3984.00
Virtualization:        VT-x
Hypervisor vendor:     Windows Subsystem for Linux
Virtualization type:   container
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse
     ↪ sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 fma cx16 xtpr
     ↪ pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave osxsave avx f16c rdrand
```

Makefile

```
#####################################################################
#####################################################################
## READ this:                                                      ##
##   https://pfzuo.github.io/2019/01/09/Install-and-Run-GPGPUSim/ ##
#####################################################################
#####################################################################

SHELL := /bin/bash # Use bash syntax
CUR_DIR = $(shell pwd)

all: benchmarks gpusim

##############################
# Make rules for gcc version #
##############################

check-gcc:
    @gcc -v 2>&1| grep -q 'gcc version 4.4' && echo 'gcc version OK' \
      || (echo 'gcc-4.4 is required' && exit 1)

    @g++ -v 2>&1| grep -q 'gcc version 4.4' && echo 'g++ version OK' \
      || (echo 'g++-4.4 is required' && exit 1)

# From https://pfzuo.github.io/2019/01/09/Install-and-Run-GPGPUSim/:
#   sudo update-alternatives --config gcc
#   sudo update-alternatives --config g++

##############################
# Make rules for gpusim tool #
##############################

gpusim: .dummy check-gcc

.dummy: gpgpu-sim_distribution/.dummy gpu-cache.cc gpu-cache.h
ifndef CUDA_INSTALL_PATH
    $(error CUDA_INSTALL_PATH is not set, try `sudo make cuda` then set environment variables appropriately.)
```

```
endif
    @if [ ! -d ${CUDA_INSTALL_PATH} ]; then \
        echo "${CUDA_INSTALL_PATH} is not found, did you set your environment variables correctly?" | exit 1; \
    fi

    @which makedepend 2>&1| grep -q 'no makedepend in' \
     && echo 'Missing dependency: try `apt-cache search makedepend`' && exit 1 || echo ''

    @cp gpu-cache.cc gpgpu-sim_distribution/src/gpgpu-sim/gpu-cache.cc
    @cp gpu-cache.h gpgpu-sim_distribution/src/gpgpu-sim/gpu-cache.h
    @cd gpgpu-sim_distribution \
     && source setup_environment \
     && make

    touch .dummy

clean-gpusim:
    rm .dummy

############################
# Make rules for benchmarks #
############################

benchmarks: check-gcc ispass2009-benchmarks/.dummy
ifndef NVIDIA_COMPUTE_SDK_LOCATION
    $(error NVIDIA_COMPUTE_SDK_LOCATION is not set, try `sudo make cuda` then set environment variables appropriately.)
endif
    @if [ ! -d ${NVIDIA_COMPUTE_SDK_LOCATION} ]; then \
        echo "${NVIDIA_COMPUTE_SDK_LOCATION} is not found, did you set your environment variables correctly?" | exit 1; \
    fi

    @mkdir -p ${NVIDIA_COMPUTE_SDK_LOCATION}/C/common/obj 2>/dev/null \
     || (echo "No write access. Try \`sudo chown -R ${USER} ${NVIDIA_COMPUTE_SDK_LOCATION}\`" && exit 1)

    @cd ${NVIDIA_COMPUTE_SDK_LOCATION}/C \
     && make 2>&1 | tee /dev/tty | grep -q "Error" \
     && (make 2>&1 | grep -q "\-lcuda" || exit 1)

    cp /usr/include/mpi/mpi.h ispass2009-benchmarks/DG/include/
    cp /usr/include/mpi/mpi_portable_platform.h ispass2009-benchmarks/DG/include/
    @cd ispass2009-benchmarks \
     && make -f Makefile.ispass-2009

clean-benchmarks:
    @cd ispass2009-benchmarks \
     && make clean -f Makefile.ispass-2009

############################
# Make rules for git repos #
############################

repos: ispass2009-benchmarks/.dummy gpgpu-sim_distribution/.dummy

ispass2009-benchmarks/.dummy:
    @if [ ! -d "ispass2009-benchmarks" ]; then \
        git clone https://github.com/gpgpu-sim/ispass2009-benchmarks.git; \
        cd ispass2009-benchmarks \
         && git checkout 1716296b22bc78acdef08fb46a338f36f9dd4b96; \
        sed -i "s/.*boost_filesystem.*/LINKFLAGS      := -L\$$(BOOST_LIB) -lboost_filesystem\$$(BOOST_VER) -lboost_system\$$(
    ↪ BOOST_VER)/g" ${CUR_DIR}/ispass2009-benchmarks/AES/Makefile; \
        sed -i "s/CUT_EXIT.*/\/\/&/g" ${CUR_DIR}/ispass2009-benchmarks/LIB/libor.cu; \
        sed -i "s/CUT_EXIT.*/\/\/&/g" ${CUR_DIR}/ispass2009-benchmarks/LPS/laplace3d.cu; \
        sed -i "s|../../../bin/release/|../bin/release/|g" ${CUR_DIR}/ispass2009-benchmarks/STO/README.GPGPU-Sim; \
        sed -i "s|../../../bin/release/|../bin/release/|g" ${CUR_DIR}/ispass2009-benchmarks/WP/README.GPGPU-Sim; \
        sed -i "s/\$$(CFLAGS) -/-fPIC \$$(CFLAGS) -/g" ${CUR_DIR}/ispass2009-benchmarks/WP/makefile; \
    fi

    touch ispass2009-benchmarks/.dummy

gpgpu-sim_distribution/.dummy:
    @if [ ! -d "gpgpu-sim_distribution" ]; then \
        git clone https://github.com/gpgpu-sim/gpgpu-sim_distribution.git; \
        cd gpgpu-sim_distribution \
         && git checkout 7cd0edfe0cb654280c30afef89a563867d9e67ed; \
        sed -i '105ireturn false;' ${CUR_DIR}/gpgpu-sim_distribution/src/option_parser.cc; \
        sed -i '105i} catch (...) {' ${CUR_DIR}/gpgpu-sim_distribution/src/option_parser.cc; \
    fi

    touch gpgpu-sim_distribution/.dummy

clean-repos:
```

```
    rm gpgpu-sim_distribution/.dummy
    rm ispass2009-benchmarks/.dummy

############################
# Make rules for cuda tools #
############################

cuda: cuda_install/.dummy

cuda_install/.dummy:
    @mkdir -p cuda_install

    @ls /root 2>/dev/null || (echo 'Try running `sudo make cuda` instead.' && exit 1)

    @echo "Installing cudatoolkit_4.0.17"
    @echo "(Place into /usr/local/cuda, and be sure to set system environment variables afterwards."
    @if [ ! -f "cuda_install/cudatoolkit_4.0.17_linux_64_ubuntu10.10.run" ]; then \
        cd cuda_install \
         && wget http://developer.download.nvidia.com/compute/cuda/4_0/toolkit/cudatoolkit_4.0.17_linux_64_ubuntu10.10.run \
         && chmod +x ./cudatoolkit_4.0.17_linux_64_ubuntu10.10.run; \
    fi
    @cd cuda_install \
     && ./cudatoolkit_4.0.17_linux_64_ubuntu10.10.run --confirm || echo ''

    @echo "Installing gpucomputingsdk_4.0.17"
    @echo "(Place into ~/NVIDIA_GPU_Computing_SDK for your non-sudo account, e.g. /home/keelin/NVIDIA_GPU_Computing_SDK)"
    @if [ ! -f "cuda_install/gpucomputingsdk_4.0.17_linux.run" ]; then \
        cd cuda_install \
         && wget http://developer.download.nvidia.com/compute/cuda/4_0/sdk/gpucomputingsdk_4.0.17_linux.run \
         && chmod +x ./gpucomputingsdk_4.0.17_linux.run; \
    fi
    @cd cuda_install \
     && ./gpucomputingsdk_4.0.17_linux.run --confirm || echo ''

    touch cuda_install/.dummy

clean-cuda:
    rm cuda_install/.dummy
```

Run Script

```
#/bin/bash

CUR_DIR=`pwd`

GPGPUSIM_DIR=${CUR_DIR}/gpusim/gpgpu-sim_distribution
BENCHMARK_DIR=${CUR_DIR}/gpusim/ispass2009-benchmarks
CUSTOM_BENCHMARK_DIR=${CUR_DIR}/CustomBenchmarks
OUTPUT_DIR=${CUR_DIR}/output
CONFIG_DIR=${CUR_DIR}/configurations

CONFIG_FILE=${CONFIG_DIR}/gpgpusim.config
CONFIG_INTERCONNECT=${CONFIG_DIR}/config_fermi_islip.icnt
CONFIG_ENERGY_MODEL=${CONFIG_DIR}/gpuwattch_gtx480.xml

mkdir -p ${OUTPUT_DIR}

for dir in ${GPGPUSIM_DIR} ${OUTPUT_DIR} ${CONFIG_DIR}; do
    if [ ! -d "${dir}" ]; then
        echo ${dir}' not found'
        exit 1
    fi
done

if [ ! -d "${BENCHMARK_DIR}" ]; then
    cd gpusim/
    make benchmarks || exit 1
fi

cd ${GPGPUSIM_DIR}/..
make gpusim || exit 1

source ${GPGPUSIM_DIR}/setup_environment

# Testing 4 indexing techniques: (L=linear, S=simple XOR, P=psuedo random interleaving, F=fermi hash set)
for set_index_fn in F P S L; do

    # Four cache configurations per technique: (fifo32, fifo64, lru32, lru64)
```

```
    assoc=4
    shmem_size=49152
    for nsets in 32 64; do
        if [ ${nsets} -eq 64 ]; then
            assoc=6
            shmem_size=16384
        fi

        # L=LRU, F=FIFO
        for replace_policy in L F; do

            sed -i "s/-gpgpu_cache:dl1 .*/-gpgpu_cache:dl1 ${nsets}:128:${assoc},${replace_policy}:L:m:N:${set_index_fn},A
     ↪ :32:8,8/g" ${CONFIG_FILE}
            sed -i "s/-gpgpu_shmem_size .*/-gpgpu_shmem_size ${shmem_size}/g" ${CONFIG_FILE}

            ##################################
            # Perform Custom Benchmarks
            ##################################

            cd ${CUSTOM_BENCHMARK_DIR}
            BENCHMARKS=`ls -1 -F | awk '/\//' | sed 's/\///'`;

            make || exit 1

            for BMK in ${BENCHMARKS}; do
                if [ -f ${CUSTOM_BENCHMARK_DIR}/${BMK}/README.GPGPU-Sim ]; then
                    # Link configuration files
                    ln -v -s -f ${CONFIG_FILE} ${CUSTOM_BENCHMARK_DIR}/${BMK}
                    ln -v -s -f ${CONFIG_INTERCONNECT} ${CUSTOM_BENCHMARK_DIR}/${BMK}
                    ln -v -s -f ${CONFIG_ENERGY_MODEL} ${CUSTOM_BENCHMARK_DIR}/${BMK}

                    cd ${CUSTOM_BENCHMARK_DIR}/${BMK}

                    # Run benchmark
                    sh README.GPGPU-Sim 2>&1| tee ${OUTPUT_DIR}/out_${BMK}_${replace_policy}${nsets}_${set_index_fn}.txt \
                      || exit 1
                fi
            done

            ##################################
            # Perform Other Benchmarks
            ##################################

            cd ${BENCHMARK_DIR}
            BENCHMARKS=`ls -1 -F | awk '/\//' | sed 's/\///'`;

            for BMK in ${BENCHMARKS}; do
                if [ -f ${BENCHMARK_DIR}/${BMK}/README.GPGPU-Sim ]; then

                    # The DG benchmark triggered my network firewall so I'm just gonna skip it
                    if [ ! "${BMK}" == "DG" ]; then

                        # Link configuration files
                        ln -v -s -f ${CONFIG_FILE} ${BENCHMARK_DIR}/${BMK}
                        ln -v -s -f ${CONFIG_INTERCONNECT} ${BENCHMARK_DIR}/${BMK}
                        ln -v -s -f ${CONFIG_ENERGY_MODEL} ${BENCHMARK_DIR}/${BMK}

                        cd ${BENCHMARK_DIR}/${BMK}

                        # Run benchmark
                        sh README.GPGPU-Sim 2>&1| tee ${OUTPUT_DIR}/out_${BMK}_${replace_policy}${nsets}_${set_index_fn}.txt \
                          || exit 1

                    fi
                fi

            done
        done

    done
done
```

PRIS Derivations

```
m=5
d(x)=x
P(x)=x^5+x^2+1

((d(x)^32) mod P(x)) mod 2
((d(x)^31) mod P(x)) mod 2
...
((d(x)^2) mod P(x)) mod 2
((d(x)^1) mod P(x)) mod 2
1

30 10010
29 01001
28 10110
27 01011
26 10111
25 11001
24 11110
23 01111
22 10101
21 11000
20 01100
19 00110
18 00011
17 10011
16 11011
15 11111
14 11101
13 11100
12 01110
11 00111
10 10001
9  11010
8  01101
7  10100
6  01010
5  00101
4  10000
3  01000
2  00100
1  00010
0  00001

b0 = a29^a27^a26^a25^a23^a22^a18^a17^a16^a15^a14^a11^a10^a8^a5^a0
b1 = a30^a28^a27^a26^a24^a23^a19^a18^a17^a16^a15^a12^a11^a9^a6^a1
b2 = a28^a26^a24^a23^a22^a20^a19^a15^a14^a13^a12^a11^a8^a7^a5^a2
b3 = a29^a27^a25^a24^a23^a21^a20^a16^a15^a14^a13^a12^a9^a8^a6^a3
b4 = a30^a28^a26^a25^a24^a22^a21^a17^a16^a15^a14^a13^a10^a9^a7^a4

###########

m=6
d(x)=x
P(x)=x^6+x+1

((d(x)^64) mod P(x)) mod 2
((d(x)^63) mod P(x)) mod 2
..
((d(x)^2) mod P(x)) mod 2
((d(x)^1) mod P(x)) mod 2
1

59 111101
58 111111
57 111110
56 011111
55 101110
54 010111
53 101010
52 010101
51 101011
50 110100
49 011010
48 001101
47 100111
46 110010
45 011001
44 101101
```

```
43 110111
42 111010
41 011101
40 101111
39 110110
38 011011
37 101100
36 010110
35 001011
34 100100
33 010010
32 001001
31 100101
30 110011
29 111000
28 011100
27 001110
26 000111
25 100010
24 010001
23 101001
22 110101
21 111011
20 111100
19 011110
18 001111
17 100110
16 010011
15 101000
14 010100
13 001010
12 000101
11 100011
10 110000
9  011000
8  001100
7  000110
6  000011
5  100000
4  010000
3  001000
2  000100
1  000010
0  000001

b0 = a30^a26^a24^a23^a22^a21^a18^a16^a12^a11^a6^a0
b1 = a30^a27^a26^a25^a21^a19^a18^a17^a16^a13^a11^a7^a6^a1
b2 = a28^a27^a26^a22^a20^a19^a18^a17^a14^a12^a8^a7^a2
b3 = a29^a28^a27^a23^a21^a20^a19^a18^a15^a13^a9^a8^a3
b4 = a30^a29^a28^a24^a22^a21^a20^a19^a16^a14^a10^a9^a4
b5 = a30^a29^a25^a23^a22^a21^a20^a17^a15^a11^a10^a5
```

Set Index Function Code

Modified from gpu-cache.cc in gpgpu-sim source code.

```
unsigned l1d_cache_config::set_index(new_addr_type addr) const{
    unsigned set_index = m_nset; // Default to linear set index function
    unsigned lower_xor = 0;
    unsigned upper_xor = 0;

    switch(m_set_index_function){
    case FERMI_HASH_SET_FUNCTION:
        /*
        * Set Indexing function from "A Detailed GPU Cache Model Based on Reuse Distance Theory"
        * Cedric Nugteren et al.
        * ISCA 2014
        */
        if(m_nset == 32 || m_nset == 64){
            // Lower xor value is bits 7-11
            lower_xor = (addr >> m_line_sz_log2) & 0x1F;

            // Upper xor value is bits 13, 14, 15, 17, and 19
            upper_xor  = (addr & 0xE000)  >> 13; // Bits 13, 14, 15
            upper_xor |= (addr & 0x20000) >> 14; // Bit 17
            upper_xor |= (addr & 0x80000) >> 15; // Bit 19

            set_index = (lower_xor ^ upper_xor);
```

```
            // 48KB cache prepends the set_index with bit 12
            if(m_nset == 64)
                set_index |= (addr & 0x1000) >> 7;

        }else{ /* Else incorrect number of sets for the hashing function */
            assert("\nGPGPU-Sim cache configuration error: The number of sets should be "
                   "32 or 64 for the hashing set index function.\n" && 0);
        }
        break;

    case SIMPLE_XOR_SET_FUNCTION:
        if(m_nset == 32) {
            // Lower xor value is bits 7-11
            lower_xor = (addr >> m_line_sz_log2) & 0x1F;

            // Upper xor value is bits 12-16
            upper_xor = ((addr >> m_line_sz_log2) >> 5) & 0x1F;

            set_index = (lower_xor ^ upper_xor);

        } else if(m_nset == 64) {
            // Lower xor value is bits 7-12
            lower_xor = (addr >> m_line_sz_log2) & 0x3F;

            // Upper xor value is bits 13-18
            upper_xor = ((addr >> m_line_sz_log2) >> 6) & 0x3F;

            set_index = (lower_xor ^ upper_xor);

        } else {
            assert("\nGPGPU-Sim cache configuration error: The number of sets should be "
                   "32 or 64 for the hashing set index function.\n" && 0);
        }
        break;

    case PSUEDO_RANDOM_INTERLEAVED_SET_FUNCTION:
        // B(x) = (A(x) / x^m)*x^m + (A(x) mod P(x))
        // H-matrix w/ ith row (from bottom):
        //   d^i mod P(x), where d is primitive element (x)
        // for n-bits a mapping to m-bits b:
        //   H[i,j] = 1 iff a{n-i-1} is an input to XOR gate outputing b{j}
        //   (counting [i,j] from bottom right)

        #define A(bit) (( (addr >> m_line_sz_log2) >> bit) & 1)
        #define B(bit,a) ( (a) << bit)

        // Using POLY(37) = x^5 + x^2 + 1 (1 of 6 irreducible polynomials of degree 5 over GF(2))
        if(m_nset == 32) {
            set_index  = B(0, A(18) ^ A(17) ^ A(16) ^ A(15) ^ A(14) ^ A(11) ^ A(10) ^ A(8) ^ A(5) ^ A(0) );
            set_index |= B(1, A(19) ^ A(18) ^ A(17) ^ A(16) ^ A(15) ^ A(12) ^ A(11) ^ A(9) ^ A(6) ^ A(1) );
            set_index |= B(2, A(19) ^ A(15) ^ A(14) ^ A(13) ^ A(12) ^ A(11) ^ A(8)  ^ A(7) ^ A(5) ^ A(2) );
            set_index |= B(3,         A(16) ^ A(15) ^ A(14) ^ A(13) ^ A(12) ^ A(9)  ^ A(8) ^ A(6) ^ A(3) );
            set_index |= B(4,         A(17) ^ A(16) ^ A(15) ^ A(14) ^ A(13) ^ A(10) ^ A(9) ^ A(7) ^ A(4) );

        // Using POLY(67) = x^6 + x + 1 (1 of 9 irreducible polynomials of degree 6 over GF(2))
        } else if(m_nset == 64) {
            set_index  = B(0,         A(23) ^ A(22) ^ A(21) ^ A(18) ^ A(16) ^ A(12) ^ A(11) ^ A(6)  ^ A(0) );
            set_index |= B(1, A(21) ^ A(19) ^ A(18) ^ A(17) ^ A(16) ^ A(13) ^ A(11) ^ A(7)  ^ A(6)  ^ A(1) );
            set_index |= B(2, A(22) ^ A(20) ^ A(19) ^ A(18) ^ A(17) ^ A(14) ^ A(12) ^ A(8)  ^ A(7)  ^ A(2) );
            set_index |= B(3, A(23) ^ A(21) ^ A(20) ^ A(19) ^ A(18) ^ A(15) ^ A(13) ^ A(9)  ^ A(8)  ^ A(3) );
            set_index |= B(4,         A(22) ^ A(21) ^ A(20) ^ A(19) ^ A(16) ^ A(14) ^ A(10) ^ A(9)  ^ A(4) );
            set_index |= B(5,         A(23) ^ A(22) ^ A(21) ^ A(20) ^ A(17) ^ A(15) ^ A(11) ^ A(10) ^ A(5) );

        } else {
            assert("\nGPGPU-Sim cache configuration error: The number of sets should be "
                   "32 or 64 for the hashing set index function.\n" && 0);
        }
        break;

    case LINEAR_SET_FUNCTION:
        set_index = (addr >> m_line_sz_log2) & (m_nset-1);
        break;
    }

    assert((set_index < m_nset) && "\nError: Set index out of bounds. This is caused by "
           "an incorrect or unimplemented custom set index function.\n");

    return set_index;
}
```