1 Lab overview

The purpose of the lab is to help students to gain hands-on experience on program optimization in terms of cache miss rate.

1.1 Environment setup

In order to finish the lab, you need to install g++, a c++ compiler and valgrind, a set of tools for debugging and profiling programs.

```
sudo apt-get install g++
sudo apt-get install valgrind
```

In Valgrind tools family, the tool "cachegrind" [link] can be used to simulate how your program interacts with the computer's cache hierarchy. Using "cachegrind", you can find your program's cache efficiency information (e.g. cache miss rate). For example, the following command can be used to find the cache efficiency information for the program "ls".

```
valgrind --dsymutil=yes --tool=cachegrind ls
```

You should see something similar to the screenshot below.

At the same time, this command generates a output file (cachegrind.out.xxxx). The first part of the file looks like below, where you can find the information of cache size, cache line size and

number of ways of associativity for your level-one instruction cache (I1), level-one data cache (D1) and last-level cache, respectively. You may find this information useful in your lab tasks.

```
I1 cache:          65536 B, 64 B, 2-way associative
D1 cache:          65536 B, 64 B, 2-way associative
LL cache:          262144 B, 64 B, 8-way associative
```

2 Lab Tasks

2.1 Task 1 - Matrix Traverse

Program list 1 (task1.cpp) shows a C++ program. This program traverses the entire matrix and assigns an integer to each of the element according its column index.

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[]) {
  if (argc != 2)
    return 0;
  int datasize = atoi(argv[1]);
  int * array = new int[datasize * datasize];
  for (int i=0; i < datasize; i++) {
    for (int j=0; j < datasize; j++) {
      array[j*datasize+i] = i;
    }
  }
  delete[] array;
  return 0;
}
```

Program List 1

Compile the program using

```
g++ -std=c++11 -o0  task1.cpp -o traverse
```

Running the below command to check its cache performance

```
valgrind --dsymutil=yes --tool=cachegrind ./traverse 10000
```

What is the D1 miss rate for this program? Can you optimize the program so the optimized-program can have a lower D1 miss rate? (Make sure the optimized program have the same outputs of the original program if given the same inputs, in other words, you are not allowed to change the meaning of the program.) Make sure to include the following 4 items in your lab report.

1, The original D1 miss rate (screenshot)
2, The optimized program (screenshot or plain source code text)
3, The D1 miss rate of the optimized program (screenshot)
4, Explanation of your optimization.

2.2 Task 2 - Matrix transpose

The following C++ program (task2.cpp) performs a matrix transpose (from src to dst).

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[]) {
  if (argc != 2)
    return 0;
  int datasize = atoi(argv[1]);
  int * src = new int[datasize * datasize];
  int * dst = new int[datasize * datasize];
  for (int i=0; i < datasize; i++) {
    for (int j=0; j < datasize; j++) {
      dst[j*datasize+i] = src[i*datasize+j];
    }
  }
  delete[] src;
  delete[] dst;
  return 0;
}
```

Compile the program using

```
g++ -std=c++11 -o0  task2.cpp -o transpose
```

Running the below command to check its cache performance

```
valgrind --dsymutil=yes --tool=cachegrind ./transpose 10000
```

What is the D1 miss rate for this program? Can you optimize the program so the optimized-program can have a lower D1 miss rate? (Make sure the optimized program have the same outputs of the original program if given the same inputs, in other words, you are not allowed to change the meaning of the program.) Make sure to include the following 4 items in your lab report.

1, The original D1 miss rate (screenshot)
2, The optimized program (screenshot or plain source code text)
3, The D1 miss rate for the optimized program (screenshot)
4, Explanation of your optimization.