In this assignment, you will conduct some studies to understand the reuse and sharing
profiles of a set of parallel program. You will instrument these shared memory parallel programs using PIN and capture the per-thread memory access traces.
Next, you will write a couple of analysis codes (outside PIN) to understand the
reuse and sharing behavior.

PART I: Collection of traces [25 points]

I have provided four parallel programs prog1.c, prog2.c, prog3.c, prog4.c. These programs process a one-million entry integer vector in different ways.
You don't have to understand these programs.

Step#1: Download Pin 3.15 and set up. Consult instructions on course home page.

Step#2: Create a directory named CS622Assignment under pin-3.15-98253-gb56e429b1-gcc-linux/source/tools/.
Copy prog1.c, prog2.c, prog3.c, prog4.c, makefile, makefile.rules into this directory.

Step#3: Compile these programs to generate static binaries. Use the following commands.

```
gcc -O3 -static -pthread prog1.c -o prog1
gcc -O3 -static -pthread prog2.c -o prog2
gcc -O3 -static -pthread prog3.c -o prog3
gcc -O3 -static -pthread prog4.c -o prog4
```

The programs need to be run with eight threads as follows.

```
./prog1 8
./prog2 8
./prog3 8
./prog4 8
```

Step#4: Write a PIN tool named addrtrace.cpp to collect the thread-wise memory access trace
of each of these binaries. Your PIN tool must convert the x86 instruction's memory accesses to
a minimum-length sequence of machine accesses before recording. We define machine accesses in the following.
Each x86 memory access instruction can access a large number of bytes. However, in your trace
you need to break such an access down into a sequence of smaller machine accesses. A machine
access can access only 1, 2, 4, or 8 bytes of memory. A machine access cannot cross the boundary
of a memory block. For this assignment, we define the memory block size to be 64 bytes.
Let us consider an example to understand the relationship between an x86 memory access
instruction captured by PIN and the equivalent sequence of machine accesses that you need to record
in your trace. Suppose PIN has captured a memory access instruction that accesses 100 bytes of memory
starting at address 125. This access will be broken down into the following sequence of machine

accesses (in each tuple, the first one is the starting address and the second one is the size of access
in bytes): (125, 2), (127, 1), (128, 8), (136, 8), (144, 8), (152, 8), (160, 8), (168, 8), (176, 8),
(184, 8), (192, 8), (200, 8), (208, 8), (216, 8), (224, 1). Since the sequence length of the machine
accesses needs to be minimized, you must use as many eight-byte accesses as possible, then as many
four-byte accesses as possible, then as many two-byte accesses as possible, and then as many one-byte
accesses as needed while obeying the constraint that a machine access must not cross the boundary
between two memory blocks and that a machine access can touch only 1, 2, 4, or 8 bytes. Each element
of the collected trace must contain a thread id and the starting address of a machine access. There is
no need to record the size of memory touched by a machine access. You can use the INS_MemoryOperandSize
function to obtain the size of the memory accessed by an x86 memory operation. This function is discussed
in the following page:
https://software.intel.com/sites/landingpage/pintool/docs/98253/Pin/html/group__INS__BASIC__API__GEN__IA32.html

If a memory operand is read as well as written to, you must consider both separately when tracing. This should
be done exactly the same way the address tracing PIN tool discussed in the class does. These are two different
memory operations and both of them must be captured and broken down separately into machine access sequences
in the trace.

Please do not try to merge multiple accesses obtained from PIN into a single large access. Here is an example
of what I am talking about. Suppose PIN has captured a memory access instruction that accesses three bytes
of memory starting at address 125 and the next one captured by PIN accesses five bytes of memory starting at
address 128. Please do not merge these two into a single access of eight bytes starting from address 125.
Instead, you should be doing the following.

Split the first access into (125, 2), (127, 1). Split the second access into (128, 4), (132, 1). Your trace will be recording 125, 127, 128, 132 along with thread id and anything else that the assignment requires.

In summary, please treat each memory access instruction captured by PIN independently.

Please make sure to use PIN_LOCKs wherever necessary.

Compile your tool and run it with PIN as follows one program at a time.

```
../../../pin -t obj-intel64/addrtrace.so -- ./prog1 8
../../../pin -t obj-intel64/addrtrace.so -- ./prog2 8
../../../pin -t obj-intel64/addrtrace.so -- ./prog3 8
../../../pin -t obj-intel64/addrtrace.so -- ./prog4 8
```

Report the total number of machine accesses recorded in the trace for each of the four programs.

The trace of machine accesses prepared in this part needs to be used in the following parts of the assignment.

PART II: Access distance analysis [35 points]

Continue to assume a memory block size of 64 bytes. Henceforth, a machine access will be referred to simply as an access. Let us define what an access distance is. Consider an access A to a memory block B. Suppose the previous access to the same memory block was A'. The access distance of the access A is defined as the number of accesses between A' and A in the trace (including A, but excluding A').
For example, if the access trace is A', X, Y, Z, A, then the access distance for A is 4.
The first access to a memory block does not have an access distance. Your task is to plot the cumulative density function F of access distances for each of the four access traces. In other words, suppose the total number of access distances computed from a trace is N i.e., the number of accesses after excluding all first accesses to memory blocks is N. Suppose the number of access distances that are less than or equal to d is n in that trace. Therefore, F(d) = n/N. If the maximum access distance seen in the trace is D, then F(D) is one.

Write a program to compute the cumulative density function of access distances from the traces.
Submit the four cumulative density function plots, one for each trace. When preparing the cdf plots, please use a log-to-the-base-ten scale on the distance axis (this should be your x-axis). Otherwise the plot's interesting points will not come out clearly.

Note:

Access distance is an approximation of reuse distance. Reuse distance computes the number of distinct memory blocks accessed between A' and A. Thus, reuse distance tells us the capacity of the cache needed to make the access A a hit. The access distance removes the uniqueness requirement and is therefore, much easier to compute and offers only a crude upper bound on the cache capacity needed to capture all reuse distances up to a certain value. For example, suppose in the cumulative density function of a trace, F(100) is 0.9. Therefore, a fully-associative cache of capacity 100*64 bytes should be able to capture 90% of all reuses.

PART III: Access distance filtered by LRU cache [20 points]

Model a single-level 2 MB 16-way cache with 64-byte block size and LRU replacement policy. Pass each trace through the cache and collect the trace of accesses that miss in the cache. Report the

number of hits and misses for each trace. Prepare the cumulative density function of the miss trace
for each of the programs. Submit the four plots. Comment on whether the shape and nature of cumulative
density function before and after the cache differ in any noticeable way. Explain your observation.
Note that at the beginning of each trace, the cache is assumed to be empty. Also, note that the
thread id should be completely ignored in parts II and III.

PART IV: Sharing profile [20 points]

In this part, you will make use of the thread id to derive the sharing profile.
Write a program to compute the number of memory blocks that are private, or shared by two threads, or shared by three threads, ..., or shared by eight threads.
The sum of these eight should be equal to the total number of memory blocks for each
trace. Report these eight numbers for each trace in a table. The LRU cache of PART III
should be removed in this part. A memory block is said to be private if it is accessed
by exactly one thread in the entire trace. A memory block is said to be shared by
exactly n threads if it is accessed by exactly n distinct threads in the trace.

WHAT TO SUBMIT

Prepare a zip ball of your submission (PIN tool, other codes, and the report; no binary
or trace please) and mail it to cs622autumn2020@gmail.com. The report should contain
the required number of machine accesses, the plots, and the tables backed by adequate
explanations and comments. I will grade only those submissions with PDF reports. Make
sure to name the zip ball groupX.zip where X is your group number. Please use only the `zip'
utility for preparing your submission. Avoid using any other compression utilities due
to possible portability issues.