Assignment 2: Cache Flushing Timer

Due: Monday 9/06/18, Multiplication Factor: 1.2

Take the bone-headed timer from assignment 1, and update it in three ways:

1. Add new cycle-accurate wall-timer option
2. Add optional sampling of multiple timing calls (specific to CPU & Wall)
3. Add optional cache flushing

We will do this work on the machine `etl.sice.indiana.edu`, so that we can avoid heavy load and control the clock speed. Your login information for this machine will be given to you in class, see professor if you have lost it. On first login, you can change the password to any secure password OKed for IU use with the `passwd` command.

Program submission will be handled the same as in assignment #1. As before, all changes (except for setting `MHZ` in `Makefile`) must be made to `dottime.c`. To get the files you need, perform:

```
mkdir 02time_cf ; cd 02time_cf
cp ~rcwhaley/classes/02time_cf/* .   # get new framework
```

Then copy your prior dottime.c to the new machine using command similar to: Then, from your `burrow`, copy your `dottime.c` from last assignment to the new machine with a command like:

```
scp dottime.c etl.sice.indiana.edu:02time_cf/.
```

The cycle-accurate walltimer should only be used when both `USEWALL`, and `PentiumMhz` are defined, with `PentiumMhz` set to the correct frequency. You can find the frequency of `classes` via `cat /proc/cpuinfo`. The cycle accurate timer is provided in the assembly file `GetCycleCount.S`, and its prototype is `long long GetCycleCount(void);`. In order to avoid type conversion and precision problems, change your timer to `long long my_time()`, and add the function `double Click2Sec(long long clicks)`, as discussed in class (make sure these timers functions are still capable of using all timers used in assignment 1, as well as the new cycle accurate wall timer!).

In order to handle sampling and flushing, you should add two optional flags to `GetFlags`:

1. `-C <kbytes>`: causes the routine to allocate at least `<kbytes>` kilobytes of data, and moves around appropriately to discourage cache reuse, as discussed in class. If set to 0, no cache flushing should be performed (i.e., it should behave as before). The default should be 32 MB.
2. `-S <nsample>`: causes each kernel invocation to be sampled `<nsample>` times. When `USEWALL` is defined, the smallest measured time is returned, and when it is not, the median time of the sample should be returned. If set to 1 or 0, the timer should behave as before. The default should be 1.

For every routine that needs these arguments (both are integers), add them to the end of the argument list. You may find it convenient to add a new routine which calls the present `DoTime()` multiple times for sampling, and you might call this routine `DoTimes()`. You may want to define a routine, compiled differently for WALL/CPU, which takes an NT-length array of times, and returns the minimum or median value; this routine could be called `double GetGoodTime(int NT, double *times)`. During debugging (but not in the final product you hand in), you should print the sampled times, and the one you are returning, so you can be sure the appropriate value is being returned.

Questions

When you `nm` (see: `man nm`) on the various dottime object files, do they in fact have the correct timer as an undefined symbol? When running the timer, can you find the rough cache edges when running with no flushing, and does the curve match the typical curve when flushing is enabled? Does performing multiple samples of smaller problems allow you to get steadier results at lower `mflop` settings? When you do only one repetition, and use the cycle-accurate wall-timer, can you get smoother results by upping the sample size? Can you get one-invocation numbers to run at the same speed as multiple (don't freak if you can't)?