

## Assignment 2: Cache Flushing Timer

Due: Wednesday 01/19/05, 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

To get the files you need, perform:

```
cp ~whaley/teach/cis5930/ASG2/* .
cp <your dottime.c from asg1> .
make archdirs
```

You will send a completed `dottime.c` to `whaley@cs.fsu.edu` by 10AM by the due date, and I expect it to work in the framework as given, so change only this file for correctness.

The cycle-accurate walltimer should only be used when both `USEWALL`, and the appropriate `Mhz` (`PentiumMhz` for `linprog`, and `SparcMhz` for `program`) is set to the correct frequency. You can find the frequency of each machine via `cat /proc/cpuinfo` and `/usr/sbin/psrinfo -v`, respectively. The cycle accurate timer is provided in the assembly file `GetCycleCount.S`, and it's prototype is `long long GetCycleCount(void);`. In order to avoid precision problems (particularly with our shiny new cycle-accurate timer), change the timer to `long long my_time()`, and add the function `double Click2Sec(long long clicks)`, as discussed in class.

In order to handle sampling and flushing, you should add two 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 4 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.

Run the timers in each of the created subdirs on the appropriate machines (Linux: `linprog`; Solaris: `program`). 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 `mflap` 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)?