COMP 40 Laboratory: Striding through memory

September 30, 2011

In this lab, you will measure the cost of memory access by exploring very regular access patterns. Figure 1 shows a program `stride.c` which repeatedly loads from the same memory locations. It takes two command-line arguments:

- Size of a memory block in megabytes
- Distance in bytes between successive reads, called the *stride*

The purpose of this lab is to enable you to practice predicting cache behavior for very simple access patterns, then check your predictions through experiment. This practice will help you predict and explain the behavior of your image-rotation programs. This program is part of the `locality` git repoistory; you can `git clone /comp/40/git/locality`. Compile with `./compile-stride`.

This lab is to be repeated on four or five computers:

- One of the `linux` login servers
- The compute server `meteor`
- The compute server `asteroid`
- **Last**, your lab machine
- As a bonus, you can try a Linux or OSX laptop, if you have one with you

Do not try a lab machine before exploring the other three! You will find the results on the lab machines quite startling.

Because `asteroid` uses a different CPU architecture, you will have to recompile when using `asteroid`.

Getting started

To get started, you will need the course software for this assignment:

```
git clone /comp/40/git/locality
```

If you already have a clone, you will need to update it by running

```
git pull -v
```

This operation will give you files `stride.c`, `compile-stride`, and `stride-lab.txt`. You can now do the lab:

1. **Complete the 13 questions** in file `stride-lab.txt`.
2. **Give your graphs to the course staff** to be scanned and returned to you.
3. **Submit your other work** using `submit40-lab-strides`.

Some of the questions require you to run code, take measurements, and plot graphs. Each graph should be labelled with this information:

1. The name of the machine
2. The name of the CPU, found by running `cat /proc/cpuinfo`
3. The CPU clock rate, also found in `/proc/cpuinfo`
4. The size of the L2 cache, also found in `/proc/cpuinfo`
5. The amount of RAM, as reported by `free -m`
6. The block size B

To learn about the output of `free`, run `man 1 free`

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <time.h>

static char xor(const char *p, const char *limit, int stride) {
  assert(stride > 0);
  char sum = 0;
  for (int offset = 0; offset < stride; offset++)
    for (const char *s = p+offset; s < limit; s += stride)
      sum ^= *s;
  return sum;
}

volatile int sink; // keeps result from being optimized away

int main(int argc, char *argv[]) {
  if (argc != 3) {
    fprintf(stderr, "Usage: %s megabytes stride\n", argv[0]);
    exit(1);
  }
  double megabytes = atof(argv[1]);
  int stride = atoi(argv[2]);
  assert(megabytes > 0 && stride > 0);

  size_t bytes = megabytes * 1024 * 1024;
  char *p = malloc(bytes);

  if (!p) {
    if ((double)(size_t) megabytes == megabytes)
      fprintf(stderr, "%s: Cannot allocate %.2fMiB\n", argv[0], megabytes);
    else
      fprintf(stderr, "%s: Cannot allocate %dMiB\n", argv[0], (int)megabytes);
    exit(2);
  }

  clock_t start = clock();
  int iter = 500 / megabytes;
  if (iter < 5)
    iter = 5;
  for (int i = 0; i < iter; i++)
    sink = xor(p, p+bytes, stride);
  clock_t stop = clock();
  double loads = (double) bytes * iter;
  double seconds = (double)(stop - start)/(double)CLOCKS_PER_SEC;
  if ((double)(size_t) megabytes == megabytes)
    printf("%dMiB", (int) megabytes);
  else
    printf("%.2fMiB", megabytes);
  printf(" stride %d results in %5.2fns CPU time per load"
         " (total %.3fs)\n",
         stride, seconds/1e-9/loads, seconds);
  return 0;
}
```

Figure 1: Program `stride.c`