Lab 05 - Performance

Out: October 15-16, 2012

1 Introduction

The purpose of this lab is to give you an understanding of various techniques that can be used to optimize the speed of your C code.

In this lab, you will be presented with several groups of functions where each function in a group performs the same task. Two groups of functions operate on an $N \times N$ image and the last group operates on a large character array. Your job is to analyze why some functions perform their task faster or slower than the naive baseline implementation using your newfound understanding of optimization techniques.

To help you out, we have included a summary of optimization techniques in *c_optimization_notes.pdf*.

2 Assignment

2.1 Installing the Stencil

To get started, run the following command in a terminal:

```
cs033_install lab05
```

This will install the code and handouts in your *course/cs033/lab05/* directory. The file with the code you will be analyzing is called *kernels.c*. To see how long each function takes to run and a comparison with its baseline function, run `make` and then run the newly compiled `./driver`.

The driver runs each of the functions and prints some information about each. The main performance measure is *CPE* or *Cycles per Element*. If a function takes C cycles to run for an image size $N \times N$, the CPE value is $\frac{C}{N^2}$. This measure is not particularly meaningful for the character array, but can still give an idea of relative performance.

The driver outputs the following statistics:

- Dim - The number of pixels along one side of the square image the function runs on (not applicable for **Reverse**)
- Your CPEs - the CPE for this function
- Baseline CPEs - the CPE for the baseline function
- Speedup - naive CPE divided by this function's CPE

2.2 Overview of the Functions

Each group of functions has multiple implementations to accomplish the same task. The functions dealing with images assume that the dimension of the image is divisible by 16.

2.2.1 Rotate

The first group of functions all rotate a square image 90 degrees counterclockwise. `dim` is the dimension of the image, `src` is the source image, and `dst` is the resulting rotated image.

The `RIDX` macro takes the x and y coordinates and the dimension of the image, and computes the one-dimensional array index corresponding to those coordinates. It is used for the sake of simplicity.

- `naive_rotate` - iterates through pixels in row-major order
- `rotate_b16` - iterates through image in 16×16 blocks
- `rotate_u16` - iterates through pixels in row-major order but uses 16x loop unrolling
- `rotate_rec` - recursively rotates image in 16×16 blocks (as opposed to `rotate_b16`)

2.2.2 Invert

The functions in the second group invert the colors in an image.

- `naive_invert` - uses the `invert_pixel` function to compute the new color at each iteration
- `invert_fast` - uses the `invert_pixel_optimized` function to compute the new color at each iteration

2.2.3 Reverse

The functions in the last group reverse a string. Unlike the functions which manipulate images and place the result in a new array, the reverse functions reverse the string in place so that the original string is replaced with its reverse.

- `naive_reverse` - iterates through a character array swapping elements from the ends and moving inward.
- `reverse_fast` - iterates through a character array with the same algorithm but loops more efficiently.

2.3 Getting Checked Off

To get checked off, describe to a TA the efficiency characteristics of each function (i.e. why the naive solutions are not optimal and why the optimizations in the other solutions help improve the CPEs). Also comment on how the optimized solutions for rotate compare to each other.

The TA may also ask you a couple questions to check your understanding.

It may be helpful to make some notes on each function to help you remember the optimization used in each function, but this is not required.