Cameron Franke
2/21/17
High Performance Computing
Program 2

Abstract

To meet the requirements for this assignment I implemented two versions of the gradient creation program. The first executes serial and follows the same algorithm presented in lecture. The second is parallelized using pthreads and splits the problem in half so that the horizontal gradient is created at the same time as the vertical gradient. This parallelization method was able to achieve a significant speedup of 3.77x on my personal computer.

Introduction

My assignment files are split into two folders within the “Program 2” folder. “Program 2” contains the Lenna images as well as the folders “pthread” and “serial”. Within each of those two folder are the source code and makefiles for the respective implementations. In addition, the output gradient images will be placed inside of their respective implementations folder in order to maintain organization.

Parallelization Methodology

To achieve parallelism in this task I implemented a two layered partitioning system. The available processors are first split into two groups; one for the horizontal and vertical gradient. This way both images are created simultaneously. By employing this method the program spends less time waiting for file writes and keeps the processors busy. Within each processor group the image processing task is split up evenly. The processors simply use their intra-group identifier to determine which portion of the image to process. Because the second convolution step is dependant on the first convolution image being completed, the processors must synchronize before moving on the the second convolution step. The ”pthread_barrier_t” tool is used to achieve this synchronization. Each group receives two barriers, one to sync after the first convolution before the second and the other to sync after the second convolution before the image write. When the convolution is complete the processor in each group begins the task of writing the image to a file. Meanwhile the main thread is waiting to join any thread that has completed its work.

Performance Testing

- Lenna 2048 x 2048, sigma = 1.1 (average of 5 tests)
  - Serial: 1115511.6 microseconds
  - Pthreads (p = 2): 570220.8 microseconds
    - Speedup = 1.95
    - Efficiency = .975
  - Pthreads (p = 4): 306763.4 microseconds
    - Speedup = 3.63
    - Efficiency = .9075
  - Pthreads (p = 8): 295668.2 microseconds
    - Speedup = 3.77
    - Efficiency = .471

This performance could be characterized as strong scalability because the speedup achieved increases as the number of processors increases. That said, the efficiency shows a rapid drop off as processor count increases. This could be an indication that the cost of communication is high or it may be a result of a relatively small data set.