CSCI 4061 – Assignment 2: Unix Process Management Report

Christopher Bradshaw

Experiment

A general overview of the program follows. A group of files (located in the "input directory") are processed by a specified number of child processes. The work done by each process is determined by its process ID—it either converts PNGs, BMPs, GIFs ("valid images") or handles "junk files" (non png/bmp/gif). Valid images are converted to the JPG format and are placed in the specified output directory.

Code was developed and executed on a laptop running Ubuntu 14.04. Source files are compiled into a single executable, parallel_convert. BASH scripts, which use this executable, were made for testing purposes. An input directory containing 45 images (with a roughly equal file type distribution) was used. For each trial, system time was taken before and after execution. The difference between these times was recorded in a CSV file. Ten (10) trials were ran for each possible number of child processes (1-10), so 100 trials were ran in total. Once every trial completed (and all the runtime data collected), the CSV file was analyzed with Microsoft Excel. Data from these trials are found in the next section.

Data Collection

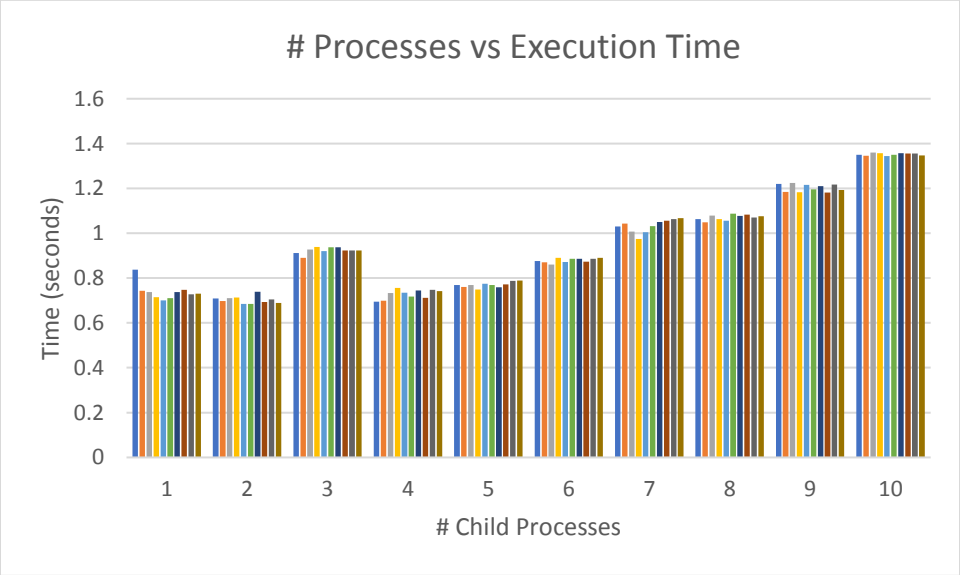

Figure 1: One hundred (100) trials of parallel_convert on an input directory with 45 images and their resulting runtime. Data points are grouped based on number of child processes (1-10) used during the trial.

# children	trial 1	trial 2	trial 3	trial 4	trial 5	trial 6	trial 7	trial 8	trial 9	trial 10	average	rank (fastest)
1	0.838058	0.743365	0.738029	0.715064	0.700489	0.709933	0.737205	0.748002	0.727167	0.730504	0.738782	3
2	0.709192	0.697993	0.711066	0.713089	0.684874	0.684587	0.738404	0.693885	0.704377	0.688683	0.702615	1
3	0.911734	0.890161	0.927429	0.938279	0.920461	0.93792	0.937934	0.923817	0.923117	0.922478	0.924622	6
4	0.69522	0.69883	0.733059	0.756018	0.734204	0.717907	0.745056	0.711672	0.748181	0.741558	0.731832	2
5	0.769506	0.760988	0.768871	0.749675	0.774296	0.76934	0.759693	0.772282	0.787419	0.788751	0.770146	4
6	0.87617	0.870363	0.86077	0.890583	0.872271	0.885605	0.885622	0.873824	0.886706	0.890947	0.879286	5
7	1.03021	1.042916	1.00805	0.974541	1.004973	1.031662	1.050684	1.055306	1.063686	1.067391	1.032942	7
8	1.063607	1.049134	1.078538	1.063578	1.056454	1.087905	1.077549	1.083015	1.06971	1.075187	1.070468	8
9	1.220308	1.184495	1.223671	1.182603	1.215897	1.195216	1.210209	1.18194	1.21788	1.192369	1.202459	9
10	1.349756	1.346349	1.360634	1.357373	1.343696	1.349628	1.357071	1.355482	1.355647	1.347064	1.35227	10

Figure 2: A more quantitative view of figure 1. Averages and their rank (lowest to highest) were calculated.

Analysis

Figure 1 shows a general upward trend in runtime as the number of child processes increases. This may be due to the way the program was implemented. Mutexes and other methods of process synchronization were not used, which could have made file selection more efficient. Without synchronization, two child processes may simultaneously select the same file to process, thus essentially wasting the processing power of a child. Synchronization was not used in this assignment because it had not yet been covered in class. Had it been used, there likely would not be an upward trend—perhaps the opposite even.

The very first trial in group 1 took longer than the other trials within the same group. This likely isn't a coincidence—there may be some overhead associated with running the program at first. Based on the computed average values, the optimal number of child processes to use is 2.