Jake Karasik, Nicholas Petriello

Time Tests

Input file = “AAAAAAAAAA”

Trials	Avg. Time 1 part (µs) process / thread	Avg. Time 2 parts (µs) process / thread	Avg. Time 3 parts (µs) process / thread	Avg. Time 4 parts (µs) process / thread	Avg. Time 5 parts (µs) process / thread	Avg. Time 6 parts (µs) process / thread
10	2743 / 2025	3904 / 3276	5169 / 4494	7053 / 5791	7754 / 6919	8759 / 8171
25	2616 / 2019	3894 / 3515	5189 / 4512	6421 / 5882	8014 / 6993	9110 / 8524
50	2654 / 2176	4026 / 3343	5489 / 4378	6723 / 5858	8086 / 7302	9087 / 8456
100	2561 / 2056	3911 / 3485	5176 / 4567	6606 / 5941	7762 / 7084	9329 / 8333

Input file = “AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEE” repeated 5 times

Trials	Avg. Time 1 part (µs) process / thread	Avg. Time 2 parts (µs) process / thread	Avg. Time 3 parts (µs) process / thread	Avg. Time 4 parts (µs) process / thread	Avg. Time 5 parts (µs) process / thread	Avg. Time 6 parts (µs) process / thread
10	2715 / 2061	4000 / 3457	5341 / 4526	6472 / 5980	7724 / 7323	9065 / 9101
25	2675 / 2049	4245 / 3377	5106 / 4748	6623 / 5818	7955 / 7502	9070 / 8716
50	2682 / 2266	3950 / 3403	5310 / 4510	6643 / 6001	7716 / 7925	9060 / 8403
100	2649 / 2167	4044 / 3402	5159 / 4574	6488 / 5982	7764 / 7060	8969 / 8425

Input file = alternating between “A” and “B” 100,000 times

Trials	Avg. Time 1 part (µs) process / thread	Avg. Time 2 parts (µs) process / thread	Avg. Time 3 parts (µs) process / thread	Avg. Time 4 parts (µs) process / thread	Avg. Time 5 parts (µs) process / thread	Avg. Time 6 parts (µs) process / thread
10	6465 / 7380	6470 / 6215	8220 / 7288	8639 / 8617	10012 / 10385	11637 / 10700
25	6155 / 7473	6255 / 6310	7600 / 7402	8819 / 8786	10336 / 9227	11549 / 11262
50	6245 / 7529	6289 / 6428	7747 / 7505	8971 / 7336	10180 / 9439	11951 / 11303
100	6255 / 7414	6228 / 6629	7854 / 7923	8852 / 8908	10045 / 10863	11881 / 11235

Input file = “AAA///BBB///CCC///” repeated 100,000 times

Trials	Avg. Time 1 part (µs) process / thread	Avg. Time 2 parts (µs) process / thread	Avg. Time 3 parts (µs) process / thread	Avg. Time 4 parts (µs) process / thread	Avg. Time 5 parts (µs) process / thread	Avg. Time 6 parts (µs) process / thread
10	47122 / 60362	26614 / 39499	22826 / 26937	25009 / 29840	23609 / 27762	22386 / 25770
25	42021 / 60213	26780 / 35121	23222 / 28126	24862 / 29424	23437 / 26723	23499 / 26516
50	41913 / 60466	26388 / 35008	22767 / 27419	26028 / 31654	23649 / 27401	22820 / 25485
100	42123 / 60459	31348 / 36925	27185 / 28051	35512 / 28839	24044 / 26611	23025 / 26418

Conclusions:

From our recorded data, it seems clear that multi-threading is faster than multi-processing almost for smaller files when using the LOLS compression algorithm. When our file size was under a few thousand characters, the threads were faster than the processes. But, when our file size was 100,000+ threading became slower. The difference in times between the first two input files barely differed even though the lengths of the files were significantly different. However, the different in times between the last two input files was significant (ex. 6465 microseconds vs. 47122 microseconds).

Jake Karasik, Nicholas Petriello

Timetest.c: Program we used to time test our code.

```c

void process_avg(int max_parts, int max_trials, long * averages) {
    int trial = 0, part = 1;

    for (;part<=max_parts;part++) {
        for(trial=0;trial < max_trials; trial++){
            long time = 0;
            struct timeval start, finish;

            gettimeofday(&start, NULL);

            compressR_LOLS("information.txt", part);

            gettimeofday(&finish, NULL);

            time += (long)(finish.tv_sec - start.tv_sec)*1000000L;
            time += (long)(finish.tv_usec - start.tv_usec);

            averages[part-1] += time;
        }
    }
    int i=0;
    for (;i<max_parts;i++){
        averages[i] /= max_trials;
        printf("process_averages %d=%ld microseconds\n",i,averages[i]);
    }
}

void thread_avg(int max_parts, int max_trials, long * averages) {
    int trial = 0, part = 1;

    for (;part<=max_parts;part++) {
        for(trial=0;trial < max_trials; trial++){
            long time = 0;
            struct timeval start, finish;

            gettimeofday(&start, NULL);

            compressT_LOLS("information.txt", part);

            gettimeofday(&finish, NULL);

            time += (long)(finish.tv_sec - start.tv_sec)*1000000L;
            time += (long)(finish.tv_usec - start.tv_usec);

            averages[part-1] += time;
        }
    }
    int i=0;
    for (;i<max_parts;i++){
        averages[i] /= max_trials;
        printf("thread_averages %d=%ld microseconds\n",i,averages[i]);
    }
}

int main(int argc, char ** argv){
    int trials=100;
    int parts = 6;
    long process_averages[parts];
    long thread_averages[parts];
    int f=0;
    for(;f<parts;f++){
        process_averages[f] = 0;
        thread_averages[f] = 0;
    }
    process_avg(parts,trials,process_averages);
    thread_avg(parts,trials,thread_averages);

    return 0;
}
```