Green Competition Report

Po-Teng Tseng

Baseline version

My baseline version has already included some optimizations I thought were good to add when I wrote it, and my optimized version has more optimizations on top of the baseline version.

Language:

I used C because I looked up related information online and read some article saying C is the most efficient language.

I/O:

I used fgets to read lines and sscanf to parse words from lines.

Data structure:

I used array of struct to store the information of word and its frequency. The struct has name (c string) and frequency (int).

Array size:

I found the array size can significantly affect the performance so I tried many different numbers. 80,000 is a good number.

Hash function:

I used a hash function called "djb2". I found it online at: http://www.cse.yorku.ca/~oz/hash.html Quadratic probing is used for solving conflict problem.

Sorting algorithm (frequency):

Quicksort should do well here as it has average O(N log N) time and it's stable. I wrote my own version of quicksort. The pivot is the median of first, middle and last element.

Sorting algorithm (alphabet):

I used insertion sort for sorting in alphabetical order, because it’s efficient for small size problem. I only sort in a group which words have same frequency.

Optimizations

Modifying quicksort function:
I tried to modify my quicksort function so it stops recursion when it’s sorting outside the range we’re interested, which increases the efficiency. Doesn’t actually help.

Combining functions:
I tried to combine my functions so the total function calls can be reduced. Doesn’t actually help.

Using qsort:
Try to use qsort in <stdlib.h> and see if it’s better. Turns out it’s much better.

Parallelism:
I used OpenMP and tried different numbers for threads. I found 5 threads has the best performance. In order to change the array into a 2d array, I have to declare the array dynamically (too large) to do that, which reduces the performance a little bit, but overall it’s very worth it.

Array size:
The array size have been changed almost every variation to see if there is a better number than 80,000 per testsuite. In my last version of code, I found it’s ok to further reduce the size as long as I reduce exact 4,000 every time. My final number is 60,000.

Results and analysis

The base line version (baseline.c) has 57.11 energy consumption .
My changes to my own functions doesn't make much difference (non parallel.c).
Dynamically declaring array consumes more energy, but it's needed for parallelism.

For non-qsort version. The energy/time consumption with different threads:

	baseline	1 Thread	2	4	5	8	14	16
Energy	57.11	61.56	49.72	51.57	46.04	51.50	45.32	47.81
Time	1.81	2.02	1.34	1.20	1.04	1.14	0.94	0.94

For qsort version. The energy/time consumption with different threads:

	1 Thread	2	4	5	8	14	16
Energy	35.80	32.06	30.84	29.91	31.92	32.50	34.86
Time	1.07	0.92	0.84	0.81	0.84	0.83	0.83

qsort out-performed my version of quicksort, and has best performance (27~29s) with 5 threads (optimized.c), and too many threads result in larger overhead and reduce performance.

My conclusion for array size is 60,000 ~ 80,000 would be a good range as long as the numbers can be divided by 4,000. The reason should be the size of the cache is fitting to these numbers.

Important findings

In this task, running qsort with 5 threads is the most energy-efficient setting I found so far.

I also learned that default functions exist for their reasons. Even though I've tried the best I can do to optimize my version of quicksort function, the built in qsort function still out-performed it very hard.

Parallelism is very useful with good number of threads.

Raw Result

Baseline results:

Program executed in about 1.810797 seconds.

About 125.5651 joules of energy were consumed, of which about 57.117 joules were due to the execution of this program.

Algorithm improved result:

Program executed in about 1.830081 seconds.

About 127.5649 joules of energy were consumed, of which about 58.3878 joules were due to the execution of this program.

hash function modified:

Program executed in about 1.821642 seconds.

About 129.1597 joules of energy were consumed, of which about 59.7853 joules were due to the execution of this program.

break for loop. dynamically declare arrays:

Program executed in about 2.015467 seconds.

About 137.7471 joules of energy were consumed, of which about 61.5624 joules were due to the execution of this program.

#pragma omp parallel

Program executed in about 0.96492 seconds.

About 89.4555 joules of energy were consumed, of which about 52.9816 joules were due to the execution of this program.

#pragma omp parallel for num_threads(2)

Program executed in about 1.342423 seconds.

About 100.4598 joules of energy were consumed, of which about 49.7163 joules were due to the execution of this program.

#pragma omp parallel for num_threads(4)

Program executed in about 1.195228 seconds.

About 96.7458 joules of energy were consumed, of which about 51.5662 joules were due to the execution of this program.

#pragma omp parallel for num_threads(5)

Program executed in about 1.03766 seconds.

About 85.2599 joules of energy were consumed, of which about 46.0364 joules were due to the execution of this program.

#pragma omp parallel for num_threads(8)

Program executed in about 1.137209 seconds.

About 94.4859 joules of energy were consumed, of which about 51.4994 joules were due to the execution of this program.

#pragma omp parallel for num_threads(14)

Program executed in about 0.941117 seconds.

About 80.8904 joules of energy were consumed, of which about 45.3162 joules were due to the execution of this program.

#pragma omp parallel for num_threads(16)

Program executed in about 0.940525 seconds.

About 83.3627 joules of energy were consumed, of which about 47.8109 joules were due to the execution of this program.

combine two frequently called functions:

Program executed in about 0.934688 seconds.

About 83.8004 joules of energy were consumed, of which about 48.4691 joules were due to the execution of this program.

Using qsort in <stdlib.h>:

Program executed in about 1.071421 seconds.

About 76.2956 joules of energy were consumed, of which about 35.7958 joules were due to the execution of this program.

qsort with #pragma omp parallel for num_threads(16)

Program executed in about 0.831626 seconds.

About 66.2975 joules of energy were consumed, of which about 34.8621 joules were due to the execution of this program.

qsort with #pragma omp parallel for num_threads(14)

Program executed in about 0.829551 seconds.

About 63.8525 joules of energy were consumed, of which about 32.4955 joules were due to the execution of this program.

qsort with #pragma omp parallel for num_threads(8)

Program executed in about 0.842798 seconds.

About 63.78 joules of energy were consumed, of which about 31.9223 joules were due to the execution of this program.

qsort with #pragma omp parallel for num_threads(5)

Program executed in about 0.813357 seconds.

About 60.657 joules of energy were consumed, of which about 29.9121 joules were due to the execution of this program.

qsort with #pragma omp parallel for num_threads(4)

Program executed in about 0.843338 seconds.

About 62.7184 joules of energy were consumed, of which about 30.8403 joules were due to the execution of this program.

qsort with #pragma omp parallel for num_threads(2)

Program executed in about 0.916688 seconds.

About 66.7128 joules of energy were consumed, of which about 32.062 joules were due to the execution of this program.