CS610 PROGRAMMING FOR PERFORMANCE
Assignment 2

September 7, 2024

SAHIL BASIA
241110061

Note

- For problem 1, I have produced results based on systems in the lab of the KD building.
- The problem 2, I have tested the code on my personal system as well as systems in the labs of the KD building.

Ans: Problem - 1

According to the problem, we must have N threads and N files. This is because each thread will open one file and read the content of that file.
The source code had some logical errors in the above scenario. I have updated the source code as this.

Commands Used:

```
>g++ modified_solution.cpp -o modified.out
>perf c2c record -o modified.data -F 30000 -u -- ./modified.out 5
./test1/input
>perf c2c report -NN -i modified.data   ./modified_rep.out
```

Similar commands can be used for original_code.

Performance Bugs:-

False Sharing on word count

To mitigate false sharing on **word count**, I moved the word count variable from shared access to the local scope of each thread. Now each thread has a separate word count variable. The earlier word count was an array and is shared among all the threads. So different threads access the different locations of the same variable/container resulting in cache invalidation(False sharing).

True sharing on lock and total word variable

Introduced padding between **total_lines_processed** and **total_words_processed**

variables to force these to variables that are present in struct word_tracker to be present on different cache lines. Padding is done to prevent any cache block invalidation due to read or write of **total_lines_processed** and **total_words_processed**. Also moved **pthread_mutex_lock(&tracker.word_count_mutex)**, **tracker.total_words_processed++**, **pthread_mutex_unlock(&tracker.word_count_mutex)** from both while loops. In simpler words, i moved these the mutex_locks, mutex_unlocks, and total_words_processed out of all the loops that are used to calculate total lines and total words. This is because, different threads were accessing the same locks and shared variables again and again i.e. same memory location again and again, which resulted in the invalidation of cache blocks.

Logical Bugs:-
1) The word count was just counting the number of space(" ") delimiters. So I modified the code and printed the word count of each thread as +1.
2) Since the number of threads are passed in the command line argument, I have implemented a check function, that checks if the number of threads passed is equal to the number of files to read. According to the question, there will be N files to read for N threads.

Output and reports
Note: The files given in the assignment had less data. So I was not able to detect any false sharing as data was sampled by perf. So I introduced some more lines in the ./test1/file*. The changed folder is shared in the problem 1 directory. I have included report files of trace of perf c2c and also included the screenshots of some outputs.

Performance Gain
1. Load Local HITM(denotes false sharing) in modified_code = 0
Local Local HITM in unmodified_code = 299
So in modified solution there is no false sharing

2. Total LOAD operations by modified_code = 23
Total LOAD operations by unmodified_code = 3496

Ans: Problem - 2

Some I/O optimization

I have done some optimization in the I/O process. Instead of repeated reading and writing into files, I have used a dynamic buffer for each. So now only one I/O operation is done for each reading the file and writing it into the file. Thus reducing the time of I/O operation.

Commands used:-

```
>g++ problem_2.cpp -o out.out
>./out.out <path to input.txt> <number of threads> <number of lines each thread reads> <size of shared buffer> <path to output.txt>
```

Example:

```
>./out.out input.txt 50 100 2 output.txt
```

Ans: # Problem - 3

for $i = 1$ to $N - 2$
  for $j = i + 1$ to N
    $A(i, j - i) = A(i, j - i - 1) - A(i + 1, j - i) + A(i - 1, i + j - 1)$

Solving A(i, j - i) & A(i, j - i - 1):-

$i_0 = i_0 + \Delta_i$
$\Delta_i = 0$ -(1)
$j_0 - i_0 = j_0 + \Delta_j - i_0 - \Delta_i - 1$
using (1)
$\Delta_j = 1$
The distance vector is $(0, 1)$ The direction vector is $(0, +)$
This has **Flow dependency**

Solving A(i , j - i) & A(i +1 , j - i):-

$i_0 = i_0 + \Delta_i + 1$
$\Delta_i = -1$ -(1)
$j_0 - i_0 = j_0 + \Delta_j - i_0 - \Delta_i$
using (1)
$\Delta_j = -1$
The distance vector is $(-1, -1)$ The direction vector is $(-, -)$

This has **Anti dependency**

Solving A(i , j - i) & A (i -1 , i +j -1):-

$i_0 = i_0 + \Delta_i - 1$
$\Delta_i = 1$ -(1)
$j_0 - i_0 = i_0 + \Delta_i + j_0 + \Delta_j - 1$
Using (1)
$\Delta_j = -2i_0$
The distance vector is $(1, -)$ The direction vector is $(+, -)$
This has **Flow dependency**