CS610: Programming For Performance

Assignment 2
Devesh Shukla (200322)

September 7, 2024

Problem 1

On analysing the given code carefully, we can see that there are three places in the program where there is cache contention. These are performance bugs and are also detectable using perf c2c tool. The tool clearly shows that there are three shared cache lines which are getting hit repeatedly under modified state (denoted as Local HITMs by the tool).

The first issue is false sharing of the elements of the shared array *word_count* between the threads. The cache line size on our system is 64 bytes and the array elements are 8 bytes each. So, 8 of them can belong to the same cache line. These get repeatedly modified multiple times by each thread during its execution contributing the most to the local HITMs detected by perf c2c. This is fixed by using an appropriate padding of 56 bytes after each array element that is actually modified by any thread. So, instead of creating an array of the same size as the number of threads, we make the array size 8 times the number of threads. And, each thread only accesses elements at index which is 8 times its id. Thus, no thread modifies the cache line used by any other thread in the program.

The second issue is cache contention because of true sharing of the variable *total_lines_processed* among all the threads. As and when each thread reads a new line, it acquires the lock on this variable and modifies it there itself. Since, the same variable is updated multiple times by each thread, it leads to several HITMs reducing program's performance. To avoid this, we create a new unshared variable *lines_processed_locally* to count the number of lines that each thread reads during its execution. This local variable is updated by each thread as it reads new lines and since, it is not modified by any other thread, no cache contention occurs. After a thread has read all lines it was supposed to read, it acquires the lock for the shared variable *total_lines_processed* and updates it. Thus, cache contention as well as contention for the lock of this shared variable is reduced drastically.

The third issue is a similar cache contention problem because of true sharing of the variable *total_words_processed* among all the threads. This is avoided same as above, by creating a new unshared variable *words_counted_locally* to count the number of words that each thread reads during its execution. This local variable is updated by each thread as it reads new words and since, it is not modified by any other thread, no cache contention occurs. After a thread has read all words it was supposed to read, it acquires the lock for the shared variable *total_words_processed* and updates it. Thus, cache contention as well as contention for the lock of this shared variable is reduced drastically this way.

For testing the performance gain, I modified the files in the sample test case to 50000 times their initial size by duplicating their original content that many times. The original program was found to take 1040 milliseconds on this test case. While after the addressing the performance bugs as stated above, the program ran in just 59 milliseconds. So, for this test case, a performance gain of nearly 17.6 times was observed. Moreover, as expected, using perf c2c for the modified program reported no cache lines with any HITMs.

Problem 2

Compilation command -

```
g++ -std=c++20 -o problem2.out problem2.cpp
```

Problem 3

Flow dependences:

Pair 1 - A(i, j - i) and A(i, j - i - 1):

$$\begin{gathered} i_o = i_o + \Delta i \\ \Delta i = 0 \\ \mathrm{j}_o - i_o = j_o + \Delta j - i_o - \Delta i - 1 \\ \Delta j = 1 \end{gathered}$$

So, direction vector = (0, +). Hence, there is a flow dependence between this pair.

Pair 2 - A(i, j - i) and A(i + 1, j - i):

$$\begin{gathered} i_o = i_o + 1 + \Delta i \\ \Delta i = -1 \\ \mathrm{j}_o - i_o = j_o + \Delta j - i_o - \Delta i \\ \Delta j = -1 \end{gathered}$$

So, direction vector = (-, -). Hence, there is no flow dependence between this pair.

Pair 3 - A(i, j - i) and A(i - 1, i + j - 1):

$$\begin{gathered} i_o = i_o - 1 + \Delta i \\ \Delta i = 1 \end{gathered}$$

$$\mathrm{j}_o - i_o = j_o + \Delta j + i_o + \Delta i - 1$$
$$\Delta j = -2i_o$$

So, direction vector = (+, -). Hence, there is a flow dependence between this pair.

Anti dependences:

Pair 1 - A(i, j - i) and A(i, j - i - 1):

$$i_o + \Delta i = i_o$$
$$\Delta i = 0$$
$$\mathrm{j}_o + \Delta j - i_o - \Delta i = j_o - i_o - 1$$
$$\Delta j = -1$$

So, direction vector = (0, -). Hence, there is no anti dependence between this pair.

Pair 2 - A(i, j - i) and A(i + 1, j - i):

$$i_o + \Delta i = i_o + 1$$
$$\Delta i = 1$$
$$\mathrm{j}_o + \Delta j - i_o - \Delta i = j_o - i_o$$
$$\Delta j = 1$$

So, direction vector = (+, +). Hence, there is an anti dependence between this pair.

Pair 3 - A(i, j - i) and A(i - 1, i + j - 1):

$$i_o + \Delta i = i_o - 1$$
$$\Delta i = -1$$
$$\mathrm{j}_o + \Delta j - i_o - \Delta i = j_o + i_o - 1$$
$$\Delta j = 2i_o - 2$$

So, direction vector = (-, +). Hence, there is no anti dependence between this pair.

Output dependences:

Writes are only happening to A(i, j - i). So, we have,

$$i_o + \Delta i = i_o$$
$$\Delta i = 0$$
$$\mathrm{j}_o + \Delta j - i_o - \Delta i = j_o - i_o$$
$$\Delta j = 0$$

So, direction vector = (0, 0). Hence, there is no output dependence.