CS3014 Concurrent Systems: Sparse Parallel Multichannel Multikernel Convolution

Conor McCauley, Sean Roche

April 6, 2020

1 Optimisations

1.1 Minor Algorithmic Changes

One minor change we made to the algorithm was to remove the initial 'zeroing' of the output matrix. The reason for this is that every element in the output matrix is already set to zero so this part of the algorithm was simply a waste of time. This change resulted in a slight decrease in execution times.

1.2 Reduction of Repeated Memory Accesses

We noticed that the `kernels` matrix was being repeatedly accessed inside of the fourth loop. We modified the code so that the repeated access is done inside of the third loop. Similarly, we noticed that the `image` matrix was being repeatedly accessed inside of the sixth loop so we modified the code so that the initial access occurs inside of the third loop and the second access occurs inside of the fourth loop. These changes resulted in a noticeable decrease in execution times.

Finally, we noticed that values in the `output` matrix was being accessed and incremented once for each index in the current kernel's `kernel_starts` interval. We replaced this access with a local variable, `sum`, which we added to the appropriate `output` value once the loop had been exited. This change resulted in a noticeable decrease in execution times for larger inputs and, strangely, a slight increase in execution times for smaller inputs.

1.3 Implementation of OpenMP

Our final and most effective improvement to the algorithm was to parallelise the bulk of our code using OpenMP. We configured OpenMP through a standard `#pragma` declaration.

We used `collapse(3)` to combine the image width and height loops as well as the first kernel loop into a single iteration space which allowed for much more multithreading and greatly reduced the execution times for large inputs.

Finally, we added an `if` condition to the declaration so that parallelisation was only utilised when the number of kernels was greater than or equal to 64. We arrived at this number of kernels through repeated trial-and-error.

2 Conclusions

Having completed this assignment we both feel like we have a deeper understanding of parallel computing and, in particular, how this can be implemented using OpenMP.

One of the challenges we faced when trying to optimise the algorithm was ensuring that the sum of the absolute differences between our output and the control output was within the given acceptable range. To do this we had to correctly privatise and share the variables within the algorithm. This took some trial-and-error but eventually we settled on sharing only the `image`, `kernels` and `output` matrices.

Another challenge we faced was dealing with the overhead introduced by setting up OpenMP. We got around this issue by introducing an `if` condition (see Â§1.3 for details).

3 Timings

Input	Average Execution Time
16 16 1 32 32 100	199 μs
64 64 3 256 256 100	134 ms
128 128 3 256 256 100	367 ms
256 256 3 256 256 100	5.01 s