**Parallel Multichannel Multikernal Convolution**

**CSU33014**

**Conor Doherty, Aaron Bruce, John Cosgrove**

*General Strategy*

For our approach, as stated in the assignment brief, we paid close attention to writing an efficient algorithm but also to other issues such as locality of data access and multiple available processor cores. Our group came together several times in order to work on our approach, with the end result being an amalgamation of optimisations suggested by each member. We made incremental changes which eventually brought our range of input sizes within the desired epsilon, while also negotiating segmentation faults and other errors along the way.

*Optimisations*

An important detail in our optimisation strategy was utilising \_m128d instructions as opposed to \_m128. It quickly became apparent that \_m128 operations resulted in values which were not precise enough and led to the sum of absolute differences being very large, even with the smallest input values. Therefore it was imperative from the offset to use operations such as \_mm\_mul\_pd instead of \_mm\_mul\_ps so ensure accuracy in all calculations.

Another optimisation we implemented was to handle vectorisation for kernel order. For kernel order 3, 5 or 7, each kernel is a 3x3 matrix and each element of the matrix is multiplied by a corresponding pixel in the image, so to vectorise this operation we can load two values from one kernel and multiply them by the two corresponding pixels. However for a kernel order of 1, each kernel only has one element. Therefore for order of 1, we load values from two separate kernels and multiply them by their corresponding pixels.

*Parallelisation*

*Conclusions*

*Timings*

|  |  |
| --- | --- |
| Input sizes | Execution times |
|  |  |
|  |  |
|  |  |
|  |  |

\_m128d to vectorise the code as \_m128(4 floats) isn’t precide enough and sum of absolute differences ends up being very large even with the smallest input values