Wessam Gholam and Eoin O'Neill

David Gregg

Concurrent Systems I (CS3014)

22 March 2017

Parallel multichannel multikernel convolution

This report discusses efforts made to modify a multichannel multikernel convolution routine to one which has parallelised and optimised whilst maintaining or improving code correctness, quality and efficiency. By adapting code to be suitable for running on a parallel processing system, assigning operations to be processed within multiple threads performance is improved. Techniques such as vectorization can be applied to transfer loops into sequences of vector operations, thereby bettering program runtime.

PROCEDURE

The initial step is to reorder the kernel. This is done so that channels can be accessed in innermost loop, meaning each channel in a pixel will be accessed rather than accessing one channel and then moving onto the next pixel. The kernel order value is then checked.

Kernel order is one. In the case that the kernel order is one, the coordinate values of the kernel (x and y) will be constant and will equal zero. This means that the x and y for loops in the original unoptimised routine can be removed for efficiency. The three outermost loops of the for loop will be parallelised using the OpenMP API. The next step is to vectorise the channels of the image and the kernel. The four different channels of the image pixel and the kernel pixel are each

loaded into two vectors. These two vectors are then multiplied and the result is stored in another vector. This vector which stores the result of the multiplication is added to a vector which keeps a sum for every iteration of the channel loop. In the case that the number of channels is not divisible by 4, a for loop iterates across all channels and horizontally sums the channel value of the remaining pixels. This is because this value would be unable to be loaded into a vector. The vector which keeps the sum of each iteration is horizontally added. This means that the adjacent elements of the vector are added together as this is an efficient way of summing together each value in the vector. The floating point value of this summation is then extracted from the vector and stored in the output array.

This process is repeated for each pixel of the kernel and the image.

Kernel order is three, five or seven. In the case the kernel order does not equal one, x and y will not be constant. This means that for this case, the for loops for each of these variables are not removed. The x, y and channel for loops are reordered again as before so that the for loop for channels is the innermost loop. This means that each channel in a pixel will be accessed before moving onto the next pixel instead of accessing one pixel and then moving onto that pixel for the next channel, which is more efficient.

From this point onwards in the program the procedure is then the same as for the case where the kernel order is 1.

RESULTS

For testing the program, each test case was ran 10 times then the average speed up of code was calculated. For all different test cases, the sum of absolute difference is within the acceptable range of Epsilon.

Test Case 1: 128 128 5 32 100

1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	Average speedup	Sum of absolute differences
91	82	83	80	85	80	80	80	87	80	82.8	~0.029460

Test Case 2: 32 32 3 64 1024

1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	Average speedup	Sum of absolute differences
87	80	81	68	67	79	88	66	68	88	77.2	~0.017420

Test Case 3: 255 255 1 63 127

1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	Average speedup	Sum of absolute differences
70	69	61	65	60	68	65	66	59	64	64.7	~0.016890

Test Case 4: 192 192 7 1 512

1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	Average speedup	Sum of absolute differences
11	13	14	14	10	11	11	14	11	13	12.2	~0.000000

CONCLUSION

This lab has illustrated real world coding experience in optimising code and speeding up programs, many optimisation techniques has been used such as vectorisation, parallelism and unrolling loops. This is displayed in the average speedup values returned for each test. The code is vectorised on the number of channels given that the higher number of channels inputted in the program implies more speed of the running time of code. One of the main challenges of the lab is to optimise code without affecting correctness of the program as it is relatively easy to optimise code and make an outcome of a program imprecise.

VERIFICATION

The code for this lab was verified by the following groups of students:

- James Mulcahy and Fiach Scolard
- Mariah Durias and Leah Shearer

Works Cited

Deshpande, Adit. "A Beginner's Guide To Understanding Convolutional Neural Networks". *Adit Deshpande - CS Undergrad at UCLA ('19)*. N.p., 2016. Web. 22 Mar. 2017.

"Intel Intrinsics Guide". *Software.intel.com*. N.p., 2017. Web. 22 Mar. 2017.

Mattson, Tim. *Introduction To Openmp: 09 Part 1 Module 5*. 2017. Web. 22 Mar. 2017.

Mattson, Tim. *Introduction To Openmp: 09 Part 2 Module 5*. 2017. Web. 22 Mar. 2017.