Parallelisation of the Block Matching Algorithm Using Three Step Search - Implementation Assignment

Daniel Bonanno (UserID: dbon0031), *University of Malta*

I. Updating the Serial Implementation

The serial implementation was updated as per the feedback received in the previous assignment. In the previous version, a vector was created and, in each index, the macroblock image, search area start and stop co-ordinates, motion vectors and Mean Square Error (MSE) were stored. The use of a vector and the data within it are no longer required since the algorithm now operates on a macroblock basis from start to end. This reduces unnecessary copying of data. The pseudocode in Figures 1 and 2 describes the old and the new methods of the serial implementation of the algorithm. As can be seen in the pseudocode, rather than subjecting every macroblock to every sequential step, storing the data along the process, all the steps are applied for one macroblock before moving on.

II. Division of the Problem and Hardware Considerations

From the pseudocode, it is clear that there is a high level of data independency: each computation performed per macroblock is independent on data found in any other macroblock. Thus, every macroblock is self-contained, in terms of data. Furthermore, for every macroblock, 27 searches are considered. In this algorithm, however, the 2nd batch of 9 search blocks is dependent on the outcome of the 1st group of 9 macroblocks and, similarly, the 3rd batch is dependent on the outcome of the 2nd. However, it can be noted that in each group, the 9 search blocks are independent from one another. Moreover, when computing the MSE between the macroblock and the search block, each pixel is considered individually. With these factors in mind, the algorithm can be split up as seen in the pseudocode in Figure 3.

```
1. Segment the image to obtain Macroblock Images
2. For Each Macroblock
      a. Obtain the Search Area Co-Ordinates
3. For Each Macroblock
      a. For 3 Times
            i.  For 9 Search Blocks
                  A. For Every Pixel in the Search Block
                        • Calculate the MSE
                  B. Obtain the MSE for Every Macroblock-Search Block Pair
                  C. Update the Motion Vector and MSE Per Block
           ii.  Update the Search Parameters
4. Reconstruct the Image
```

Figure 1:: The Pseudocode for the First Version of the Serial Code

```
1. For the number of possible macroblocks
      a. Obtain the Macroblock details which the algorithm will work on
      b. Obtain the Search Area Co-Ordinates
      c. For 3 Times
            i.  For 9 Search Blocks
                  A. For Every Pixel in the Search Block
                        • Calculate the MSE
                  B. Obtain the MSE for Every Macroblock-Search Block Pair
                  C. Update the Vector and MSE Per Block
      d. Reconstruct the part of the image that corresponds to this macroblock
```

Figure 2:: The Pseudocode for the Second Version of the Serial Code

```
1. For 3 Times
      a. Create a block for macroblock – search block pair in the image
         (therefore, 9xnumber of macroblocks GPU blocks).
      b. Create a thread for every pixel in compute the MSE for that pixel
      c. Obtain the MSE for each macroblock - search block pair
      d. Choose the Optimal MSE and Calculate the Motion Vector
      e. Update the Search Parameters
```

Figure 3: Pseudocode for the Parallel Implementation

Every macroblock therefore, has 9 blocks. In order to distinguish between the different search blocks, they are enumerated as shown in Figure 4. The same enumeration is present in the code when referring to search blocks.

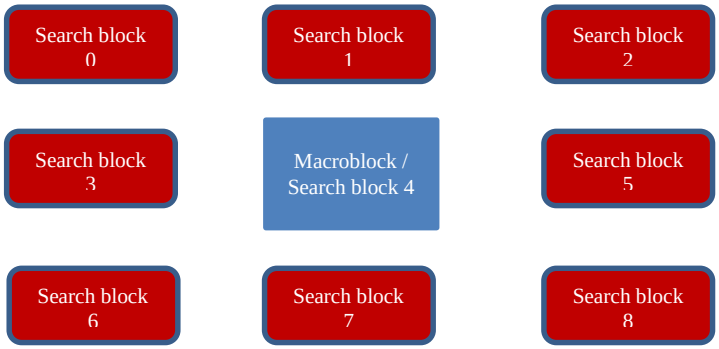

Figure 4: The numbers assigned to each search block belonging to each macroblock

From the above implementation, it is clear that the number of GPU blocks and threads created is dependent on the resolution of the image used and the block size. Given that it the maximum number of threads in a block is 1024 and that the number of threads created is determined by the user-specified block size, an additional precaution needs to be taken such that the number of pixels in a block is not larger than 1024.

Also, the implementation ensures that threads in the same block will share certain parameters: indexing parameters such as the macroblock index and search area co-ordinates as well as the MSE value for a macroblock – search block pair. Thus, these variables are stored in shared memory. The MSE for the macroblock – search block pair is the sum of all the values generated by the threads. Thus, all the threads write their result to an array in shared memory.

It can also be noted that the images passed to the device are linearized, as shown in Figure 5. The linearization process was designed such that memory is coalesced. This will be discussed further in the next section.

Apart from obtaining the MSE values, the general structure of the program remains the same as that of the amended serial implementation. The parallel algorithm also generates the same results as the serial implementation. This was verified by running both the serial and parallel implementations various times: using frame 1 as both the reference and the frame to be reconstructed, using 2 different frames, using different resolutions and using different block sizes. In all the cases, both results were subtracted from one another. The result of the subtraction is 0 in every case, meaning the images generated are identical.

III. Advanced Facilities Used

NB: All profiler runs were obtained using a Titan Black GPU and using the release version of the code.

A. Thread Granularity and Latency Hiding

One of the main features of programming on a GPU is the fact that thread creation is relatively inexpensive when compared to thread creation on the host. Thus, creating many threads is encouraged, especially when considering the fact that an instruction executed by threads in a warp might have to wait for a long-latency operation to finish. In this case, the GPU can switch to another warp whilst waiting out the latency period. To this aim, the number of threads created, although dependent on the user-specified block size, is as large as possible. Instead of creating a thread per macroblock-search block pair, as with other parallel interpretations of this algorithm [1], a thread per pixel is created. Figure 6 shown below notes how the occupancy for a 20x20 block is approximately 81%, which is quite good. It can be increased if larger block sizes are used, as shown by the graph in Figure 7.

Moreover, threads in the same block are capable of sharing memory. Such memory is faster than global memory. As described above, given that threads in the same block operate on the same macroblock, they are able to share data such as the macroblock index as well as the array where the result are stored. This allows the

Figure 5: The linearization process of a 3 channel image

implementation make use of the hierarchical structure of the memory in a GPU. As can be seen in Figure 8, most memory reads and writes are from shared memory, which is faster than global memory.

Variable	Achieved	Theoretical	Device Limit
Occupancy Per SM			
Active Blocks		4	16
Active Warps	51.71	52	64
Active Threads		1664	2048
Occupancy	80.8%	81.2%	100%

Figure 6: The occupancy for a 20x20 block

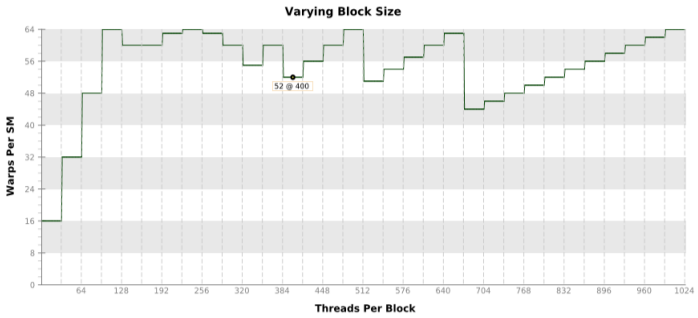

Figure 7: The occupancy in terms of warps per SM as the number of threads per block changes.

Transactions	Bandwidth	Utilization
L1/Shared Memory		
Local Loads	73	982.082 kB/s
Local Stores	68	410.788 kB/s
Shared Loads	9404600	256.558 GB/s
Shared Stores	9096363	248.149 GB/s
Global Loads	7840813	69.313 GB/s
Global Stores	21182	62.633 MB/s
Atomic	10	10 B/s
L1/Shared Total	26363109	574.084 GB/s
L2 Cache		
L1 Reads	20326583	69.314 GB/s
L1 Writes	21250	72.463 MB/s
Texture Reads	0	0 B/s
Noncoherent Reads	0	0 B/s
Atomic	20	34.1 kB/s
Total	20347833	69.386 GB/s
System Memory		
[PCIe configuration: Gen2 x16, 5 Gbit/s]		
Reads	15	134.054 kB/s
Writes	216	1.93 MB/s

Figure 8: Most loads and stores are performed on shared memory, which is faster than global memory

B. *Coalesced Memory*

In order to maximize memory bandwidth, threads in half a warp should read consecutive memory addresses. This is true for the implementation of the algorithm. Each thread will need to read pixel values. When linearizing the image to pass it the kernel, it is ensured that adjacent pixel values are placed next to each other. This ensures that threads in half a warp read adjacent memory addresses, reducing memory-read latency. In fact, considering the stall reasons as per Figure 9 below, it is clear that memory dependency is quite low.

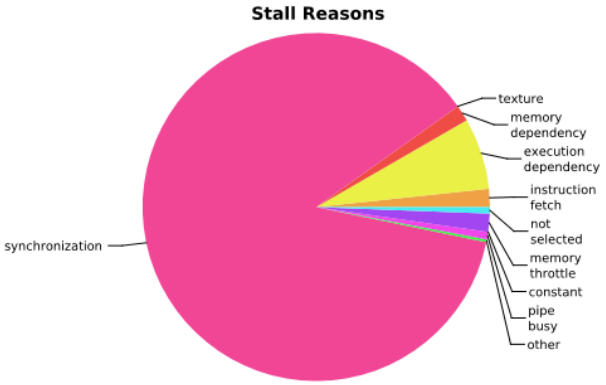

Figure 9: Stall Reasons for the current kernel

C. *Divergence*

Since a block is created for every macroblock – search block pair, there is no need to check if a pixel MSE should be calculated or not due to the pixel being out of range. Either all the block threads need to compute the MSE or none at all! This therefore, avoids divergence when it comes to the threads.

Some divergence is necessary only when setting up the initial values and summing up, since only one thread should be allowed to write to certain variables at a time. This results in the largest proportion of latency, as shown in Figure 9.

D. *Floating Point Considerations*

The MSE must be calculated as a float, since it might be very small (<0) and rounding errors may result in loss in precision. Having said that, a 32-bit float is adequate, rather than a 64-bit double, which would offer more

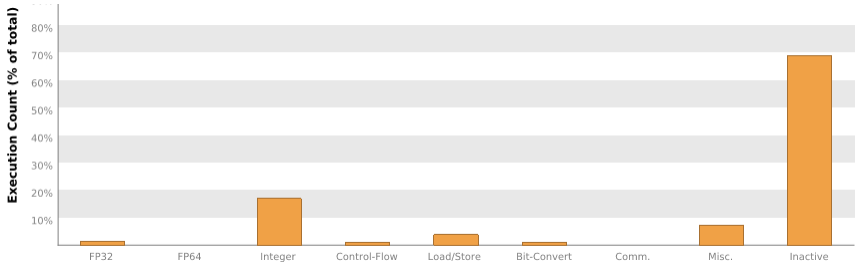

Figure 10: The execution count, grouped by classes

precision at the cost of a larger computation time. Also, other variables which are integers are declared as such, ensuring that integer arithmetic is used, rather than having them declared as a float or double and rounded. With that in mind, it can be noted (Figure 10) that most operations are integer operations and 32-bit floating point operations are kept to a minimum, whilst 64-bit floating point operations are non-existent. The inactive threads are due to synchronization, as discussed above.

IV. Optimization of the Code

A. *Synchronization*

Clearly, the largest problem with the implementation is the fact that a lot of threads are inactive whilst waiting for the initial setup of parameters. With this in mind, it was noted that the pixel level MSE array can be initialized by all the threads rather than just 1 thread. With this optimization, the algorithm executes faster, since this removed serial nested for loops and made better used of the GPU threads available. In fact, observing the new stall reasons in Figure 11, stalling due to synchronization has been reduced drastically.

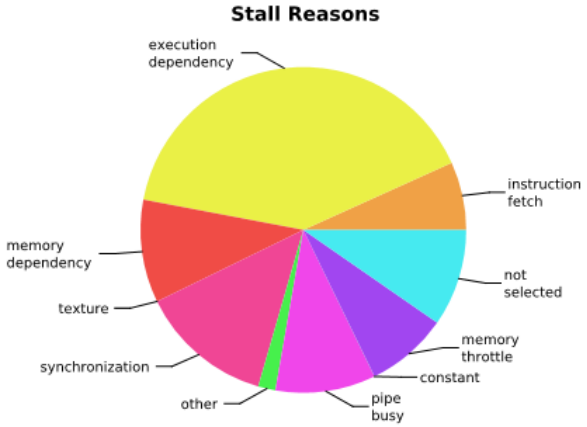

Figure 11: An updated version of the stall reasons

B. *Use of Pinned Memory*

In order to increase the memory bandwidth, pinned memory is used, such that certain data is page-locked. Thus, the host would no longer need to get to first get the data from pageable memory, put it into pinned memory and then transfer it to the device, but rather put it in the device immediately. Using this type of memory, the memory bandwidth was increased, as shown in Figure 12. It can also be noted that read and write times for System Memory were also increased.

C. *Overall Improvement*

The overall improvement of the optimizations can be observed in the tables of results below.

Transactions	Bandwidth	Utilization
L1/Shared Memory		
Local Loads	47	1.644 MB/s
Local Stores	46	857.946 kB/s
Shared Loads	10288950	735.614 GB/s
Shared Stores	2192660	156.765 GB/s
Global Loads	10077561	187.968 GB/s
Global Stores	129816	1.158 GB/s
Atomic	5	5 B/s
L1/Shared Total	22689085	1,081.508 GB/s
L2 Cache		
L1 Reads	21032861	187.97 GB/s
L1 Writes	129756	1.16 GB/s
Texture Reads	0	0 B/s
Noncoherent Reads	0	0 B/s
Atomic	10	44.684 kB/s
Total	21162617	189.129 GB/s
System Memory		
[PCIe configuration: Gen2 x16, 5 Gbit/s]		
Reads	15	134.054 kB/s
Writes	216	1.93 MB/s

Figure 12: The new memory utilisation values

V. Results

NB: All tests were run on using a Tesla K40 GPU and using the release version of the code.

Both the parallel and the serial implementations were tested for different resolutions and block sizes, with the result tabulated below in Tables 1 and 2.
For the results regarding the block size, the resolution is kept at a constant 1280x720p, whilst for the different resolutions test, the block size was set to a constant 20x20. Clearly, the speedup varies according to the parameters set by the user.

For calculating a theoretical speedup, in this case, Amdahl's Law is not considered, since the speedup measured here is not for the whole application but rather for the algorithm. With that in mind, the following factors are taken into consideration: the CPU speed, the speed of each GPU core and the number of GPU cores present. With these, the following equation can be formed:

$$Speedup = \frac{GPU\ Clock\ Speed \times Number\ of\ GPU\ Processors}{CPU\ Clock\ Speed}$$

As per the specifications provided, the CPU speed is 2.80GHz, the GPU clock speed for the Tesla K40 is 0.75GHz and the Tesla K40 has 15 Multiprocessors, each having 192 CUDA cores. Thus, a total of 2880 CUDA cores are available. This means that a theoretical speed up of approximately 771.43 times is possible.

VI. Difficulties in Achieving Full Utilization

It can be observed that the algorithm itself is highly parallelizable, since data dependency is very high. Having

said that, the algorithm parallelization is limited by the fact that the 2nd and 3rd steps of the Three Step Search are depended on the 1st and 2nd steps respectively. This means that the search must be performed in a serial manner. Also, due to hardware limitations, the number of pixels present in a block is limited. Since in this case, the threads per GPU block is equal to the size of the macroblock, this means that macroblock sizes are limited. Another speedup might have been achieved by parallelizing the addition of the pixel-level MSE to obtain the MSE for the search block and macroblock pair. This would mean that, instead of adding numbers in the array serially, threads could be used. Having said that, the size of the array is limited by the size of the macroblock, which is limited to 1024 pixels due to the hardware. Therefore, although a speedup might be present, it may also be minimal.

The operation that takes the most time is the copy of data from host to device. However, this is necessary and cannot be avoided. In this case, the bandwidth for the transfer was improved with the introduction of pinned memory. Also, the thread synchronization present in the beginning of the kernel execution is required, since it initializes the shared block variables. This cannot be avoided, although it causes threads to become inactive in a warp.

REFERENCES

[1] E. Monteiro, B. Vizzotto, C. Diniz, B. Zatt and S. Bampi, "Applying CUDA Architecture to Accelerate Full Search Block Matching Algorithm for High Performance Motion Estimation in Video Encoding", 2011 23rd International Symposium on Computer Architecture and High Performance Computing, 2011.

Table 1: Results for different block sizes, resolution is kept at a constant 1280x720

Block Size (pixels)	Serial Time (s)	Parallel Time (s)	Parallel Time Optimised (s)	Speedup Factor
4x4	4.6693	0.1677	0.1447	32.27
8x8	2.2221	0.1453	0.1375	16.16
20x20	1.0810	0.1506	0.1216	8.890

Table 2: Results for Different Resolutions, block size is kept at a constant 8x8

Resolution	Serial Time (s)	Parallel Time (s)	Parallel Time Optimised (s)	Speedup Factor
640x360	0.5321	0.1147	0.0996	5.32
848x480	1.0370	0.1318	0.1128	9.19
1280x720	2.2092	0.1421	0.1375	16.07
1920x1080	4.87486	0.1871	0.1735	28.10