SDP Memo 092: The FFT calculation via NVIDIA cuFFT library

Document number……SDP Memo 092
Document Type……...….MEMO
Revision……...1
Author……...Karel Adámek, Wes Armour
Release Date……..……...….2018-10-24
Document Classification……..…… Unrestricted

Lead Author	Designation	Affiliation
Karel Adámek		Oxford e-Research Centre, University of Oxford
Signature & Date: Karel Adámek Karel Adámek (Oct 30, 2018)		

SDP Memo Disclaimer

The SDP memos are designed to allow the quick recording of investigations and research done by members of the SDP. They are also designed to raise questions about parts of the SDP design or SDP process. The contents of a memo may be the opinion of the author, not the whole of the SDP.

Table of Contents

Abstract

We present measured execution time and performance in FLOP/s of the NVIDIA cuFFT library for 1D and 2D Complex to Real FFT calculation. We have examined wide range of FFT lengths including lengths which are powers of two, these are favorable to FFT algorithms, but also lengths which are non-power of two. The cuFFT performance is limited by device memory bandwidth. The compute utilisation of the GPU varies but it is below 16%. The effect of PCIe bus, e.g. time required to transfer data from host to the GPU, was not included.

Keywords — predict, SKA, GPU

1 Results

We present measured performance and execution time of 1D and 2D complex to real FFT using the cuFFT library. The GPU used was the TITAN V from the Volta generation of NVIDIA GPUs. Technical specifications of this card are listed in Table 1. We have not included time required to transfer data via PCIe from the host to the GPU for processing and back. We have also not covered numerical precision analysis as it was covered by S. Salvini [2].

The cuFFT library uses different implementations of the FFT algorithm depending on FFT length and also on type of FFT performed. In case of complex-to-real transformation it is better to use out-of-place method, because it is faster.

Table 1: GPU card specifications. The shared memory bandwidth is calculated as BW(bytes/s) = (bankbandwidth(bytes)) × (clock frequency (Hz)) × (32 banks) × (# multiprocessors).

	TITAN V
Total CUDA Cores	5120
Peak performance	15.7 TFLOP/s
Streaming Multiprocessors (SMs)	80
Base/Max Core Clock	1220/1455 MHz
Memory Clock	850 MHz
Global memory bandwidth	652 GB/s
Shared memory bandwidth	14550 GB/s
Global memory size	12 GB
TDP	250 W
CUDA version	9.1.85
Driver version	387.34

1.1 1D FFT

The execution time per one FFT, achieved number of FLOPSs and compute utilisation of the GPU for 1D FFT calculation for both single and double precision is shown in Figure 1. For the measurements we have used FFT lengths N which are powers of two as well as length which are not. We have also performed longest FFT possible given the limitations of GPU's device memory size. The FFT algorithm performs best when N is powers of two. For other FFT lengths the performance drops significantly which we see clearly in the performance plot in Figure 1 on the left. For the performance measured in TFLOP/s we have used computational complexity

$$O(\text{1DFFT}) = 5N \log_2(N). \tag{1}$$

In case of double precision we assume that one operation performed on numbers in double precision takes two FLOPs. This is due to GPU architecture which performs double precision operation using two single precision units. Using double precision also increases amount of data which needs to be moved for FFT of the same size.

In figure 1 on the right we also present execution time for both single and double precision. The double precision FFT's takes roughly twice as much as time to perform than single precision FFT. We also see that non-power of two FFT lengths might take much longer time to compute then FFT sizes which are powers of two. It is usually better to pad the FFT with zeros (or mean) to the nearest higher power of two length, because in spite of longer FFT we get the result faster. For some FFT lengths this is shown in Table 2. However padding itself, especially when multiple FFT are padded, could be expensive as well and it also means increasing data size for all subsequent algorithms which will process data further.

In Figure 2, we also present execution time required to process fixed amount of data

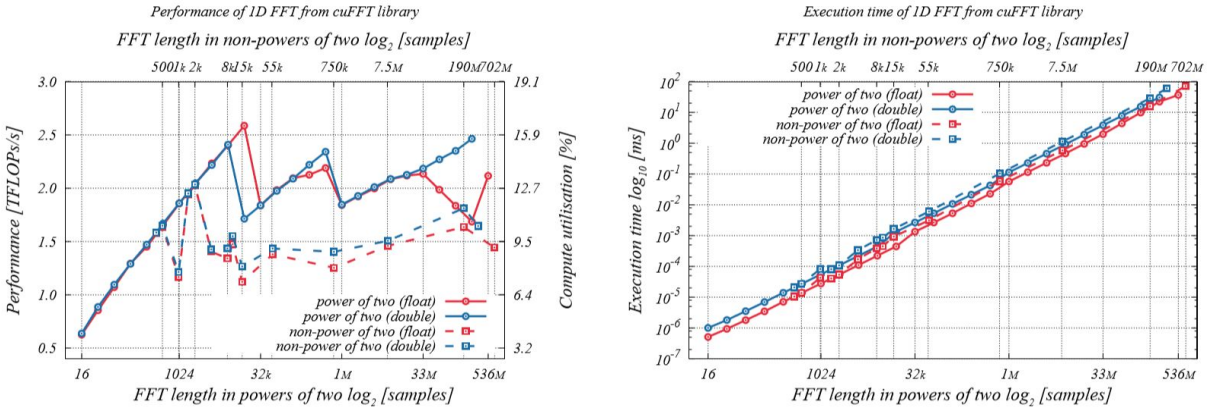

Figure 1: Achieved performance [TFLOP/s] on the left and execution time [ms] on the right for both single and double precision for 1D FFT calculation using cuFFT library. The performance of the non-power of two FFT lengths is significantly smaller then for the power of two FFT lengths. The drops in performance for power of two FFT lengths shows where cuFFT uses a different algorithm to calculate the FFTs. In execution time we see that non-power of two FFTs takes much longer to calculate then FFTs with power of two length.

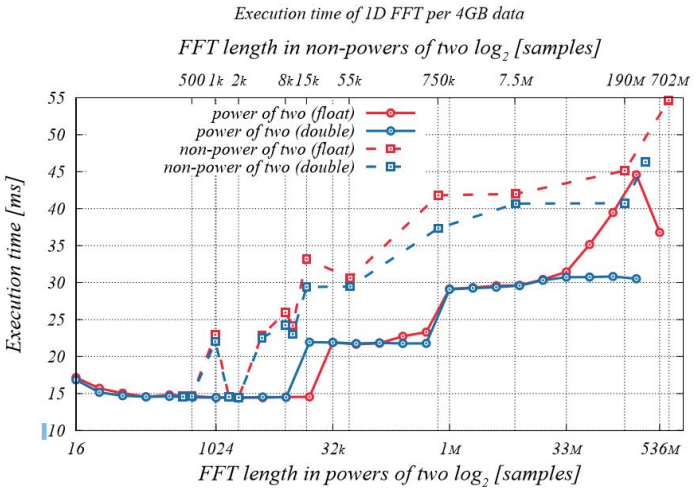

Figure 2: Execution time[ms] of 4GB of data for single and double precision for 1D FFT calculation using cuFFT library. With increasing FFT length less FFTs are performed.

Table 2: Comparison of execution times (speed-up) of non-power of two FFT length vs. padded power of two FFT length for 1D FFTs in single precision. The speed-up is given as (ex. time of non-power of two)/(ex. time of power of two). Time required for padding is not included.

FFT length	Speed-up
1000 (1024)	1.53×
2000 (2048)	0.98×
3000 (4096)	1.60×
4000 (4096)	1.53×
8000 (8192)	1.65×
10000 (16384)	1.01×
15000 (16384)	2.07×

(4GB). This might be useful since it eliminates the size of the input data and it shows efficiency of different kinds of implementations of the FFT algorithm used by the cuFFT library. It could also help in estimation execution time of cuFFT library, knowing amount of data which needs to be processed. The jumps in execution time (if we look only at powers of two FFTs) signifies use of a different algorithm by the cuFFT library. The 1D FFT is completely limited be device memory bandwidth. The NVIDIA SKA memo [1] points out that older generations of NVIDIA GPUs had problems utilising full memory bandwidth due to latencies for texture load instructions. The NVIDIA memo [1] also mentions that this limitation will be alleviated in further generations. This seems to be true as device memory bandwidth is utilised up to 90% of the theoretical peak.

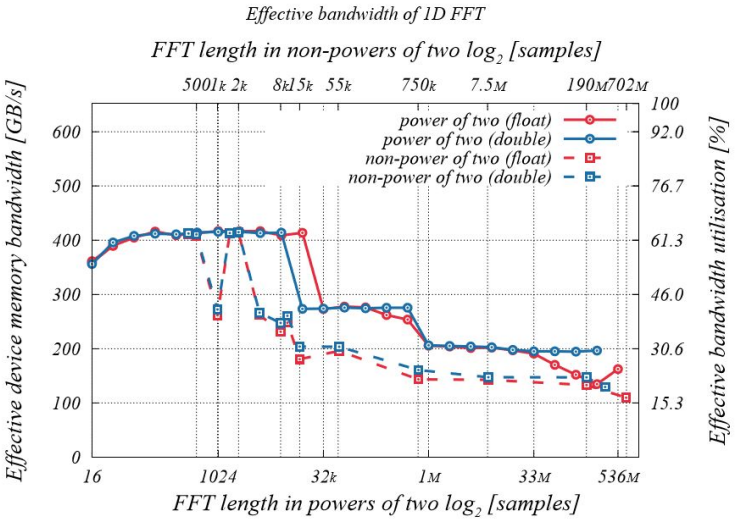

Figure 3: Effective bandwidth for 1D FFTs using cuFFT library. The drops are caused by the change of the algorithm used for calculation of FFTs.

Lastly we present effective bandwidth, shown for 1D FFT in Figure 3. The effective bandwidth assumes that the algorithm have ideal data reuse, that is the input is read from and output written to memory only once. This is not the case for FFT algorithm. For long FFTs

the FFT algorithm needs to write to and read from device memory multiple times since every element of the output of the FFT algorithm is a weighted sum of all elements of the input. The effective bandwidth for given FFT length can help us estimate execution time required to process FFT of given length knowing peak device memory bandwidth. Low effective bandwidth does not mean that device memory bandwidth is not a limiting factor as demonstrated by FFTs, where long FFTs might have low effective bandwidth but at each step cuFFT library is limited by device memory bandwidth and utilization of the device memory bandwidth is very high. In case of 1D FFF, long FFTs require more frequent access to device memory and thus it has lower effective bandwidth.

1.2 2D FFT

For the 2D FFTs of size $N \times N$, the execution time, achieved number of FLOPs and compute utilisation of the GPU is shown in Figure 4. As in 1D case we have used combination of power of two and non-power of two FFT lengths N. The cuFFT library using decomposition of 2D FFT into two series of 1D FFTs, one in each dimension. This approach requires transpose of the data which further increases pressure on device memory bandwidth. Some of these transfers might be eliminated depending on implementation. For computational complexity of 2D FFT we have used same formula as in [2] to allow comparisons, that is

$$O(\text{2DFFT}) = 5N^2 \log_2(N). \tag{2}$$

We see again that FFTs of non-power of two size have greatly reduced performance and as in 1D case most times it is better to pad data to nearest higher power of two. The execution time of double precision FFTs compared to single precision FFTs is same as in 1D case, that is, it takes twice as long to compute. This is more due to because of data size rather then compute utilisation because 2D FFT are again completely limited by device memory bandwidth. The table 3 shown speed-up which we get if we have used nearest power of two FFT length instead of non-power of two length.

As in 1D FFT case we present execution time for 4GB of input data, this is shown in Figure 5. We also present effective bandwidth for 2d FFTs in Figure 6. The cuFFT library uses the separability of 2D FFTs into two sets of 1D FFT in each direction with matrix transpose in between, thus the effective bandwidth for 2D FFTs is much lower as these additional operations adds much more memory accesses.

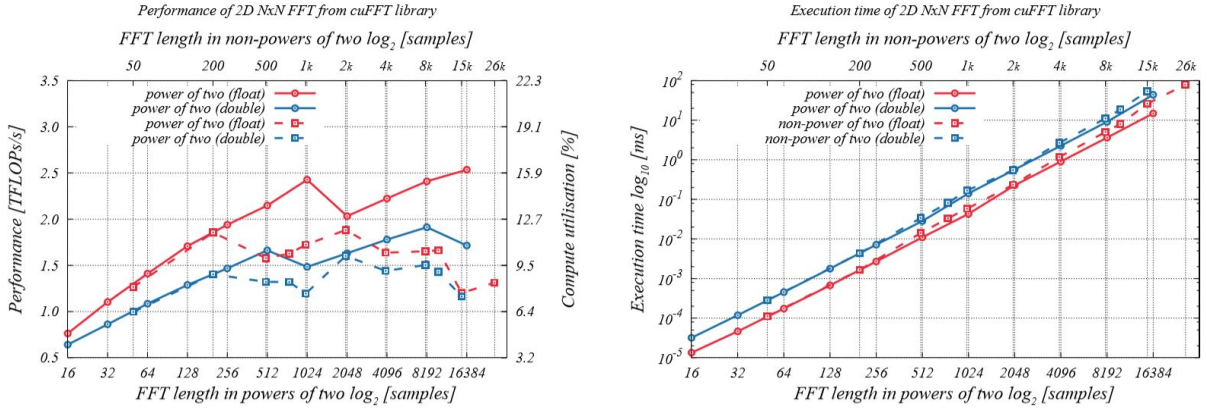

Figure 4: Achieved performance [TFLOP/s] on the left and execution time [ms] on the right for both single and double precision for 2D FFT calculation using cuFFT library.

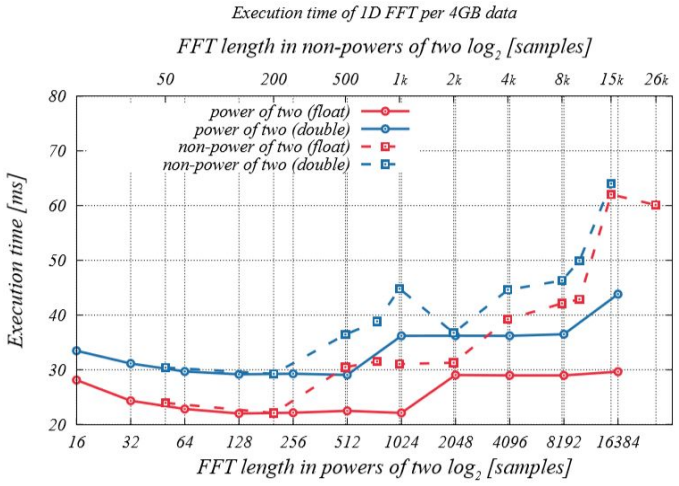

Figure 5: Execution time[ms] of 4GB of data for single and double precision for 1D FFT calculation using cuFFT library. With increasing FFT length less FFTs are performed.

Table 3: Comparison of execution times (speed-up) of non-power of two FFT length vs. padded power of two FFT length for 2D FFTs in double precision. The speed-up is given as (ex. time of non-power of two)/(ex. time of power of two). Time required for padding is not included.

FFT length	Speed-up
1000 (1024)	1.32×
2000 (2048)	1.02×
3000 (4096)	1.04×
4000 (4096)	1.28×
8000 (8192)	1.39×
10000 (16384)	0.54×
15000 (16384)	1.74×

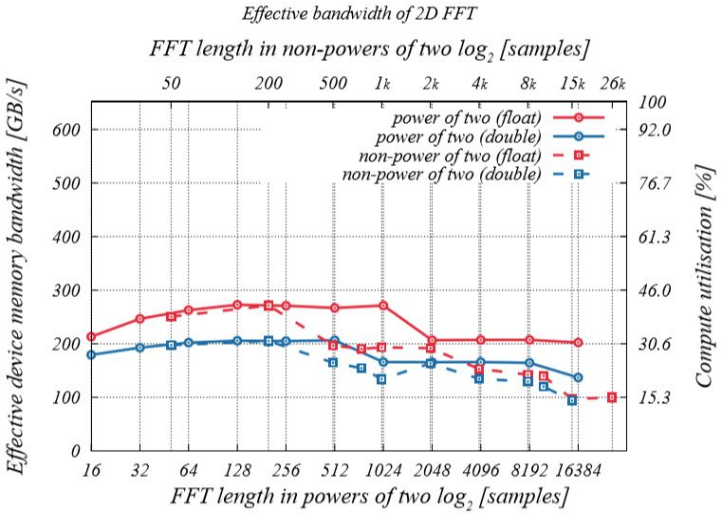

Figure 6: Effective bandwidth for 2D FFTs using cuFFT library. The 2D FFTs has lower effective bandwidth because in addition to the FFTs, cuFFT library has to transpose 2D data as well.

2 Conclusion

The performance of the cuFFT library for in 1D and 2D FFT is completely limited by device memory bandwidth. The compute utilisation of GPU depends on FFT length and algorithm used by the cuFFT library, but it is mostly below 15%. The FFTs lengths which are non-power of two should be avoided, by padding the data with zeroes (or mean) to the nearest power of two FFT length. However this might not be applicable in all cases since padding itself has associated cost with it and it increases data size for further processing. Each such case should be investigated more closely. Smaller FFT lengths should be processed in batches otherwise GPU would be under-utilised and perform poorly.

The performance of the cuFFT library was investigated for SKA purposes two times, once by NVIDIA [1] and once by S. Salvini [2]. Both investigations used NVIDIA K40 GPU from Kepler generation. When compared to NVIDIA TITAN V GPU from Volta generation used for this work, the K40 device memory bandwidth was 288GB/s and peak floating point performance was 5040 GFLOP/s. The TITAN V has device memory bandwidth 652GB/s (2.26× more) and peak floating point performance of 15700 GFLOP/s (3.1× more). Since cuFFT library is limited by device memory bandwidth we would expect that TITAN V would be 2.2× faster then K40 in computations of FFTs. This is indeed what we see when we have compared our results with results from S. Salvini [2]. The average speed-up for single precision is 2.6× and average speed-up for double precision is 2.3×. There are improvements in performance for specific FFT lengths. In general non-power of two FFT lengths have higher then expected performance when compared to K40. This is mostly due to improvements in algorithms rather then hardware. We also see improvements for longer power of two FFT lengths, for example in single precision with speed-up 3.0× for FFT length 16384. This is again due to algorithm improvements in CUDA 9.0 package.

We have presented two additional metrics which could help to estimate execution time of cuFFT library for given FFT lengths. First is execution time of the cuFFT library on fixed amount of data (4GB) which could help to estimate execution time when data throughput is known. The second is effective bandwidth which could be used to estimate execution time of the cuFFT library when device memory bandwidth is known.

References

[1] NVIDIA. Characterizing fft performance on gpus for ska-sdp, 2016.

[2] S. Salvini. Sdp memo: Fast fourier transforms. http://ska-sdp.org/publications/released-sdp-memos, 2015.