SDP Memo 076: Improving the efficiency of direct visibility prediction using NVIDIA Pascal and Volta GPUs

Document Number ..... SDP Memo 076
Document Type ..... MEMO
Revision 1
Author ..... Karel Adámek, Fred Dulwich
Release Date 2018-09-12
Document Classification ..... Unrestricted
Status ..... Draft

Lead Author	Designation	Affiliation
Karel Adámek		Oxford e-Research Centre, University of Oxford
Signature & Date: Karel Adámek Karel Adámek (Oct 24, 2018)		

Revision	Date of issue	Prepared by	Comments
1	2018-09-12	Karel Adámek, Fred Dulwich	First draft

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

A visibility prediction stage may be required multiple times throughout an SKA data processing pipeline. In this report we outline a method for direct visibility prediction using GPUs, and describe the optimisations we used to improve the efficiency of the algorithm on these GPUs. We present measured execution time and performance in FLOPs, as well as measured speed-ups over the original code used for the predict step in the OSKAR interferometer simulator. We measured the performance of four different versions of the predict algorithm, each of differing complexity, and each in single and double precision. Significant performance improvements were possible in seven out of the total of eight different versions. The algorithms were benchmarked using two NVIDIA GPU cards: the P100, which is a scientific GPU card based on the Pascal architecture; and the TITAN V, which is a GPU card based on the newer Volta architecture. Because of architectural differences between the two generations of hardware, the best optimisation strategy depended on the type of GPU being used, as well as the complexity of the algorithm. The achieved speed-ups generally increase further as the number of components in the sky model grows larger.

1 Introduction

1.1 Scope of this work

Predicted visibilities can be generated either by direct evaluation of a Fourier sum over discrete sky model components (using an explicit Measurement Equation, Hamaker et al., 1996), or by taking the fast Fourier transform of a model sky image, and then de-gridding the result onto the projected interferometer baselines. The direct-evaluation method is very flexible as it allows various effects to be modelled accurately for sources at arbitrary distance from the phase centre, although it is computationally intensive as the number of operations scales linearly with the number of sky model components and the number of baselines. The second method may be more appropriate to use for highly structured images of sky models around the phase centre. In practice, both methods are typically employed in radio astronomy pipelines, depending on the requirements of the data processing stage and the complexity of the sky model in use.

In this report we consider only the direct-evaluation version of predict, as implemented by the OSKAR interferometer simulator (among others). More details of the predict step in OSKAR and how it may differ from the predict step employed for telescopes like the SKA can be found in Appendix A. The predict step is likely to be required in multiple stages of a data processing pipeline, and the computational complexity of each stage depends on the effects it needs to evaluate. Based on this, we investigated four different versions of the direct prediction algorithm which differ in computational complexity, by including or excluding polarization, time smearing and bandwidth smearing effects, all in both single and double precision. Where accurate model visibilities are required for sources far from the phase centre, for example in calibration, a predict step using full Jones matrices with time and/or bandwidth smearing might be appropriate. In the image deconvolution process it may be sufficient for the predict step to consider Stokes I only, without smearing effects, for sources close to the phase centre.

1.2 Roofline for GPUs

For this work we used two NVIDIA GPU cards: a P100 based on the Pascal architecture, and a TITAN V based on the newer Volta architecture. The P100 is a scientific GPU card designed for HPC environments, while the TITAN V is a high-end consumer gaming card, but it is the only representative of the Volta architecture currently available to us[1]. The hardware specifications of these devices can be found in Table 1.

Table 1: GPU card specifications. The shared memory bandwidth is calculated as $\mathrm{BW}(\mathrm{bytes/s}) = (\mathrm{bank\,bandwidth\,(bytes)}) \times (\mathrm{clock\,frequency\,(Hz)}) \times (32\,\mathrm{banks}) \times (\#\mathrm{multiprocessors})$.

	P100	**TITAN V**
Total CUDA Cores	3584	5120
Streaming Multiprocessors (SMs)	56	80
Base/Max Core Clock	1126/1303 MHz	1220/1455 MHz
Memory Clock	1406 MHz	850 MHz
Global memory bandwidth	720 GB/s	652 GB/s
Shared memory bandwidth	9121 GB/s	14550 GB/s
Global memory size	16 GB	12 GB
TDP	250 W	250 W
CUDA version	8.0.44	9.1.85
Driver version	384.81	387.34

The Roofline model (Williams et al., 2009) is an abstract model used to understand the performance limitations of a piece of software on a given hardware platform. It allows us to guess whether an algorithm will be bandwidth bound or compute bound, and gives an indication of how problematic the implementation of an algorithm could be on given hardware. For example, the Roofline model indicates if we need to develop cache-optimised code, or if we should rather consider optimisations in compute. We have constructed the Roofline graph for both GPU cards available to us (P100, TITAN V); these are shown in Figure 1 and in Figure 2. The Roofline model combines operational intensity (OI; the number of floating point operations performed by the algorithm per byte of data transferred) on the x-axis, with memory bandwidth, in the form of a slanted line, and indicates the achievable FLOPs. The prominent feature of the Roofline model is the ridge point, where the slanted line (the roof) changes to a horizontal line (the ceiling). The position of the ridge point tells us the number of floating point operations we need to perform per byte in order to fully utilise the hardware, given the memory bandwidth limitation. If we examine the Roofline graph for the GPU cards we used, considering global device memory bandwidth only, we see that the ridge point for the P100 is at operational intensity $\mathrm{OI} = 14$, and for the TITAN V it is at $\mathrm{OI} = 24$ (assuming single precision operations). This is equivalent to performing 56 or 96 single precision operations per floating point number. These values of operational intensity are very high, and are unlikely to be achieved for simple implementations of most algorithms. So without reusing data in caches or in shared memory, it is very hard to achieve full hardware utilization. When we plot a Roofline graph using shared memory bandwidth instead, we can achieve full compute utilisation at operational intensity $\mathrm{OI} = 1.16$ for the P100 and $\mathrm{OI} = 1.08$ for the TITAN V. These values of operational intensity for shared memory bandwidth show the importance of cache (shared memory) utilisation on GPUs and the need for data re-use if we want to reach a higher compute utilisation.

[1] The Volta generation GPU card designed for HPC environments is called V100.

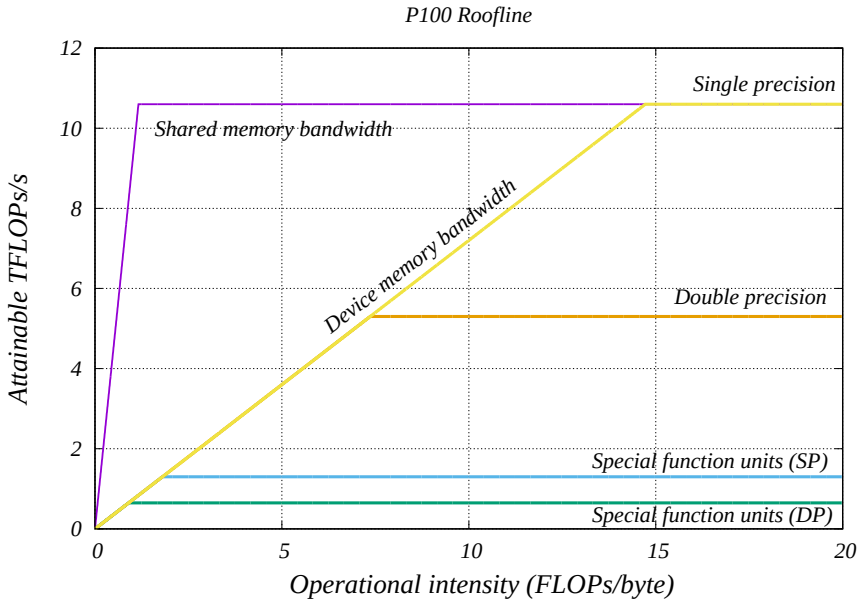

Figure 1: Roofline model for the P100 GPU. The ridge point for global device memory is at $\mathrm{OI} = 14.72$, but for shared memory it is located at $\mathrm{OI} = 1.16$. It shows that without data re-use, it is very hard to fully utilise GPU computational potential.

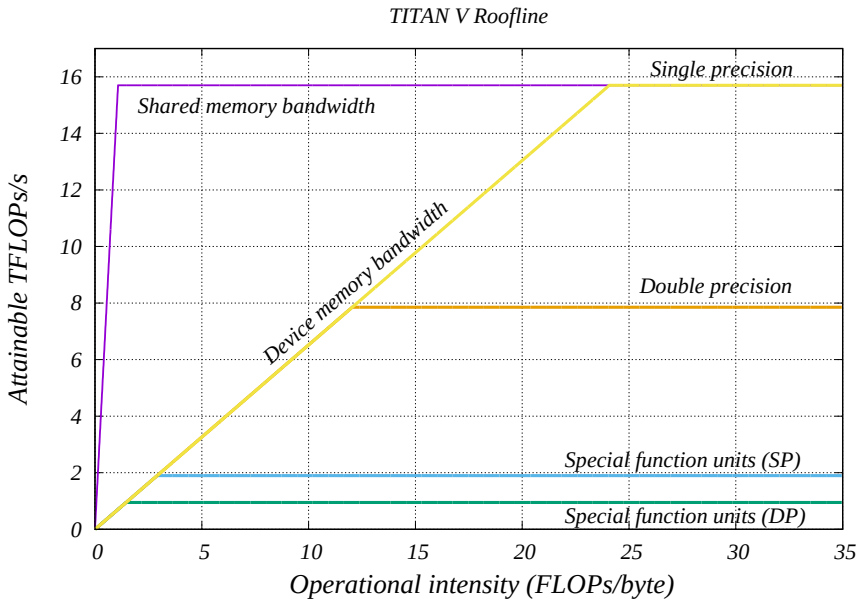

Figure 2: Roofline model for the TITAN V GPU. The ridge point for global device memory is at $\mathrm{OI} = 24.07$, but for shared memory it is located at $\mathrm{OI} = 1.08$. It shows that without data re-use, it is very hard to fully utilise GPU computational potential.

1.3 Visibility prediction using direct summation

The output from a predict stage is a set of model visibilities on each baseline for a given sky model and telescope configuration. In the case of a predict step which includes polarization, the visibility matrix $V_{\mathrm{p,q}}$ for a baseline between stations p and q for s sources is given by a Measurement Equation (e.g. Smirnov, 2011)

$$V_{\mathrm{p,q}} = \sum_s V_{\mathrm{p,q,s}} = \sum_s \zeta_\tau(p,q,s) \cdot \zeta_\nu(p,q,s) \cdot \mathbf{J}_{\mathrm{p,s}} \cdot \mathbf{B}_{\mathrm{s}} \cdot \mathbf{J}^H_{\mathrm{q,s}}, \tag{1}$$

where ζ_τ and ζ_ν are time and bandwidth smearing terms respectively, $\mathbf{J}_{\mathrm{p,s}}$ is the Jones matrix for station p and source s and $\mathbf{B}_{\mathrm{s}}$ is the brightness matrix of the source. This variant of the predict step is the most computationally complex case we investigated, requiring a total of 23 real numbers (8 per Jones matrix, 4 Stokes parameters and 3 direction cosines) and performing 136 floating point operations per source and one $\mathrm{sinc}()$ special function call per smearing term. The operational intensity is then $\mathrm{OI} = 1.48$ in single precision. If smearing terms are not required, the number of variables decreases to 20 and the number of operations to 122 FLOPS. This increases operational intensity to $\mathrm{OI} = 1.53$ and eliminates the need for special function units.

The Jones matrices are pre-calculated for each source and station and are read from global memory. The time and bandwidth smearing corrections depend on both the baseline and the source, so these are computed on-the-fly using one $\mathrm{sinc}(x)$ function for each.

The visibilities generated using Equation 1 include effects which might not always be necessary to evaluate, such as the smearing terms, or in many cases it might be sufficient to consider Stokes-I visibilities only, where Jones matrices could be replaced by complex scalars. These simplifications would save computational or memory bandwidth resources and improve the performance of the predict step.

The simplest scalar case can be deduced from Equation 1 by reducing the 2x2 Jones matrices to complex scalars. The total visibility is given by

$$V_{\mathrm{p,q}} = \sum_s V_{\mathrm{p,q,s}} = \sum_s \zeta_\tau(p,q,s) \cdot \zeta_\nu(p,q,s) \cdot K_{\mathrm{p,s}} \cdot I_{\mathrm{s}} \cdot K^*_{\mathrm{q,s}}, \tag{2}$$

where $K_{\mathrm{p,s}} = \exp\left(-2\pi i \left(u_p l_s + v_p m_s + w_p (n_s - 1)\right)\right)$ is a complex scalar. Since $K_{\mathrm{p,s}}$ are pre-calculated for each source and station and are loaded from scratch memory on the device, the total real numbers required per source is 8, or 5 without smearing terms. The number of floating point operations is 28, or 14 without smearing terms; thus the operational intensity in single precision is $\mathrm{OI} = 0.88$ or $\mathrm{OI} = 0.7$ respectively. If we choose to include smearing terms we need one $\mathrm{sinc}()$ function call per smearing term, which are calculated by special function units.

The operational intensity for the predict step is highest for the case where we include polarization without smearing terms with value $\mathrm{OI} = 1.53$. This operational intensity is well below the ridge point for both GPU cards we investigated. It also suggests that it is best to focus our attention on optimising data re-use in caches.

1.4 The baseline code

The direct-evaluation form of the predict algorithm is naturally parallelisable, as each contribution from a source s to a baseline p,q is independent of contributions from other sources to that baseline, and baselines are also independent from each other. The original code in OSKAR exploits this by performing the summation in multiple stages, and is described as follows: each

thread-block calculates the visibility on a single baseline p,q from all sources; each thread within the thread-block loops through independent subsets of sources and accumulates their contributions into a visibility matrix local to the thread; after summing all assigned sources, a thread writes its local visibility matrix into shared memory; finally, the total visibility for the given baseline is updated by summing all local visibilities, as given by Equation 1. The final step is performed by a single thread. The pseudo code for the original version in OSKAR is shown in Figure 3 on the left.

Algorithm 1: original predict

```
// station indices
p =blockIdx.x;
q =blockIdx.y;

// avoid duplicating baselines
if q ≥p then return;

// set up pointers to Jones
   matrices
// all threads handle same
   baseline
st_p = &Jones [p];
st_q = &Jones [q];

for all sources do
    // neighbouring threads
       within block access
       neighbouring sources

    accumulate local visibilities;
end
store local visibilities to shared
  memory;

if threadIdx.x ==0 then
    calculate total visibility for
      baseline p,q;
    write out total visibility to global
      memory;
end
```

Algorithm 2: optimised predict

```
// station indices
p =blockIdx.x;
q =blockIdx.y*BPK + w;

// avoid duplicating baselines
if q*BPK≥p then return;

// set up pointers to Jones
   matrices
// different warps handle
   different baselines
st_p = &Jones [p];
st_q = &Jones [q*BPK + w];

for all sources do
    __syncthreads();
    // neighbouring threads
       within warp access
       neighbouring sources
    // different warps access
       same sources
    accumulate local visibilities;
end
total visibility calculated
within warp;

if i==0 then

    write out total visibility to global
      memory;
end
```

Figure 3: Comparison between the original OSKAR predict code (left) with the cache-optimised version (right). The variable BPK is number of baselines per thread-block, which depends on the number of threads per block as $\mathrm{BPK} = \#\mathrm{threads}/32$, w is the warp ID calculated as $w = \mathrm{threadIdx.x}/32$, and i is the local thread ID within the warp, calculated as $i = \mathrm{threadIdx.x}\ mod\ 32$. The shared-memory-optimised version differs only by storing data into shared memory at the beginning of each iteration of the source loop.

Table 2: Summary of metrics from the NVIDIA visual profiler for the original OSKAR predict and the shared memory optimised version of the predict code for 5000 sources running on the P100.

Metric	single		double	
	original	shared mem.	original	shared mem.
Memory utilisation [%]	95 (L2)	65	85 (L2)	75 (L2)
Compute utilisation [%]	55	85	35	55
Occupancy [%]	74.1	74.5	37.2	31.2
Occupancy theoretical [%]	75.0	75.0	37.5	31.2
#Registers per thread	39	40	80	88
Shared mem. bandwidth [GB/s]	505 (6%)	3393 (37%)	350 (4%)	4609 (50%)
L2 cache bandwidth [GB/s]	1396	867	1138	1008
Unified mem. bandwidth [GB/s]	5210	4062	3608	2866
Global mem. bandwidth [GB/s]	411 (56%)	148 (20%)	467 (64%)	176 (24%)

Table 3: Summary of metrics from the NVIDIA visual profiler for the original OSKAR predict and the cache optimised version of the predict code for 5000 sources running on the TITAN V.

Metric	single		double	
	original	cache	original	cache
Memory utilisation [%]	75 (Gl.)	35	75 (Gl.)	45
Compute utilisation [%]	35	75	35	65
Occupancy [%]	73.8	62	48.4	43.6
Occupancy theoretical [%]	75.0	62.5	50.0	43.8
#Registers per thread	37	44	62	68
Shared mem. bandwidth [GB/s]	532 (4%)	1034 (7%)	326 (2%)	536 (4%)
L2 cache bandwidth [GB/s]	1531	1245	1457	1529
Unified mem. bandwidth [GB/s]	6950	13603	4839	8883
Global mem. bandwidth [GB/s]	520 (80%)	256 (39%)	519 (80%)	281 (43%)

For a small number of sources (5000), the original version of the code running on a P100 is limited by L2 cache. According to the NVIDIA visual profiler, the P100 L2 cache is utilised up to 95% for single and 85% for double precision, whilst the compute utilisation is 55% in single precision and only 35% in double precision. This suggests that data are already re-used in cache, which is supported by medium utilisation of global memory bandwidth. On the other hand, the TITAN V is limited by global memory bandwidth. The TITAN V utilisation levels in both single and double precision are a global memory bandwidth of 75% and compute of 35%, so data are re-used less. Using a larger number of sources, the original code in OSKAR quickly becomes limited by global memory bandwidth. These and other metrics are summarised for both the original OSKAR predict and our optimised code in Table 2 and Table 3.

The OSKAR predict code is also limited by the number of registers (variables) it uses per thread. If the number of registers used by a thread is too large, not all of the GPU's resources can be utilised and the number of active, running threads in a given instance decreases. This is especially relevant in the double precision case where the number of registers in use is roughly doubled.

2 Optimization strategy

One way to improve the performance of memory-bound code is to exploit opportunities for more data re-use. For this algorithm, data re-use can be increased by changing the way baselines are handled. Each baseline is defined by two antennas p and q. We can create new baselines by using the same antenna p while changing antenna q, taking care to not duplicate baselines by using the same antennas p and q in reverse order. This allows the code to retain 15 out of 23 variables (65% of data) per source.

We have implemented two optimised versions of the direct predict algorithm. Both versions fix one of the stations in order to re-use data in a more optimal way. We use the same distribution of work as in the original version, but we increase the number of baselines processed per thread-block. To do this we use the natural division of threads into warps[2], where each warp calculates the total visibility matrix for its own baseline, i.e. uses a different station q. This way, threads from different warps access the same sources. (This is in contrast to the original method in OSKAR where each thread from the thread-block accesses a different source, so although memory accesses could be coalesced, there was no data re-use within the thread-block.) When different warps process the same sources their accesses to memory are consolidated, which helps to ensure that data are still in the cache when requested. This was effective for a small number of sources but failed for more then 20k sources. To further ensure that data would be re-used we have added a synchronization step into the source loop, which forces all warps from the thread-block to use the same sources at the same time. The comparison between the original OSKAR version and this cache optimised version is shown in Figure 3.

In order to decrease L2 cache load we also implemented a shared memory version of the algorithm. Instead of relying on cache, we can store the data required for the current iteration of the source loop into shared memory. Each source is loaded from cache once and then re-used from shared memory multiple times. Since the size of the shared memory is very limited, we are able to store enough data only for one iteration of the source loop.

To reduce number of registers per thread we tried to calculate only one of four elements of the visibility matrix. This reduces number of variables needed per thread from 31 (23 for input

[2] A warp is a group of 32 threads which execute instructions in lock-step.

and 8 for output) to 17 (15 for input and 2 for output). However, initial tests did not lead to the desired performance gain. Further investigation might reveal opportunities for additional performance increases.

3 Results

We tested both the cache and the shared-memory versions on both GPUs. Each GPU performed better with different implementations: The cache-optimised version was generally better suited to the TITAN V, while the shared-memory-optimised version generally performed better on the P100. All presented results are for 512 stations, or 130,816 baselines.

3.1 Polarised, smeared visibilities

The results presented in this section focus on the version using full 2x2 Jones matrices, with time and bandwidth smearing. The performance results in the form of speed-ups over the original version are shown in Figure 4, and the execution time of the new versions is shown in Figure 5.

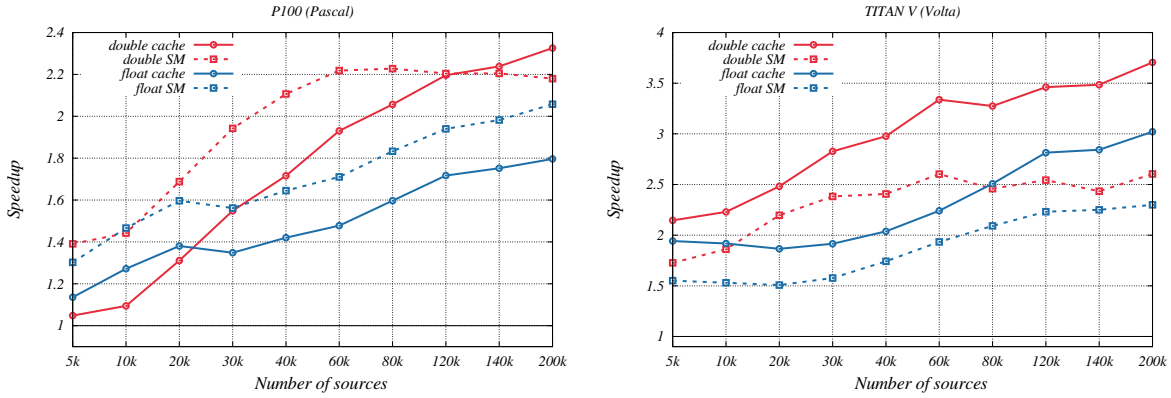

Figure 4: Performance results of both codes on the P100 (left) and the TITAN V (right) in both single and double precision. For the P100, the cache predict does achieve higher speed-ups for 120k and larger source counts.

We have successfully increased the performance of predict on the P100, which is limited by L2 cache, by $1.3\times$ for single precision and $1.4\times$ for double precision. A significant performance increase was also achieved in the case of TITAN V, where single precision for 5000 sources is $1.9\times$ faster and double precision $2.1\times$ faster than the original OSKAR predict. The higher speed-ups for the TITAN V could be expected since it starts from being global memory bandwidth bound instead of being L2 cache bandwidth bound as it is in the case of the P100. For a larger number of sources the performance gain increases.

For a small number of sources ($\leq 20k$), the optimised version (with the exception of double precision on the P100) is compute bound by the special function units, which calculate $\mathrm{sinc}()$ functions. With a larger number of sources the code becomes global memory bandwidth bound. This suggests that dividing larger numbers of sources into subsets and processing them separately should improve performance.

Lastly we present the change in performance introduced by adding a synchronization step into the cache-optimised predict code. This is most relevant for TITAN V rather than for P100.

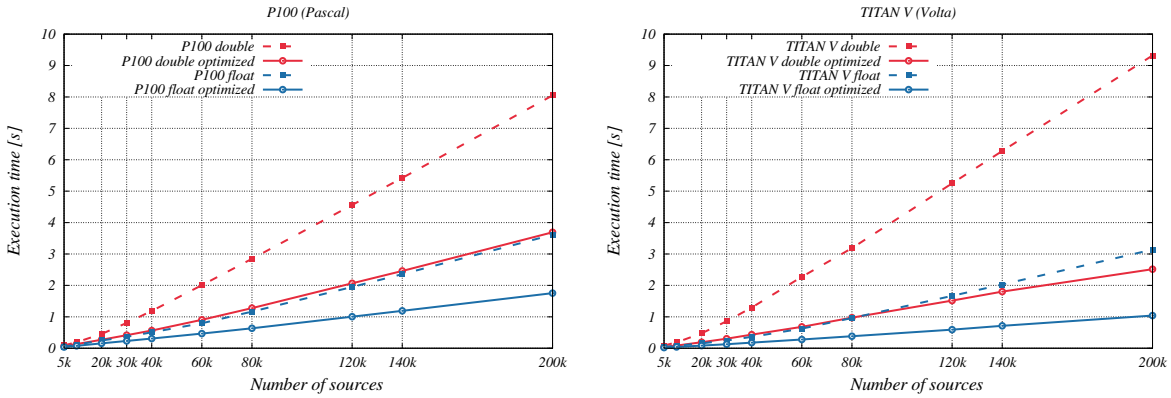

Figure 5: Execution time for both single and double precision on the P100 and the TITAN V. It should be noted that the original OSKAR predict in double precision is slower on the TITAN V in spite of the TITAN V being a faster card. This is because on the TITAN V, the code is limited by global memory bandwidth and not L2 cache as on the P100.

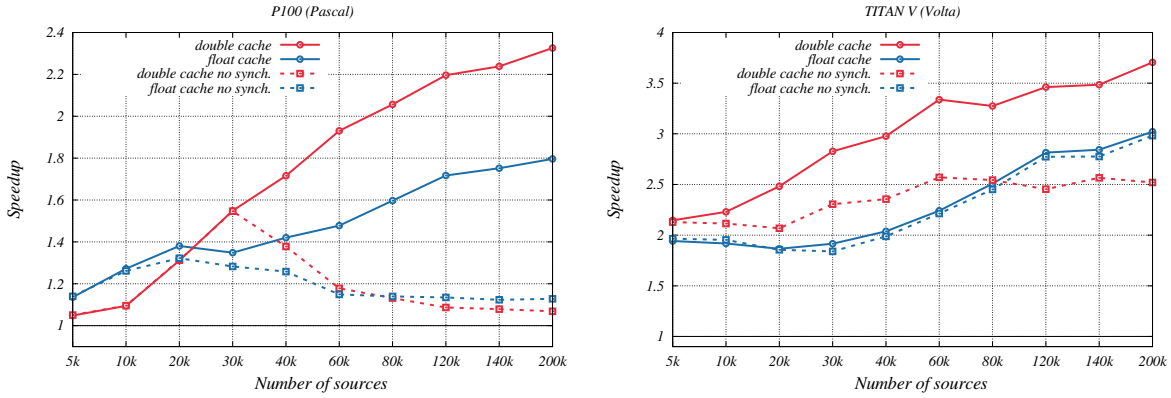

Figure 6: Effect of adding a synchronisation step into the cache optimised code. The TITAN V is less sensitive to adding synchronisation and performance remains good for single precision. In double precision the effect is more pronounced.

This is shown in Figure 6. The P100 is very sensitive to the synchronization step and data re-use markedly improves after 30k sources. A similar effect can be observed for TITAN V as well, but only for double precision. For 200k sources the performance on the TITAN V is reduced by 40%.

3.2 Comparison of different modes

In this section we present results to compare the performance improvements only for the kernel itself in different modes: with and without polarisation; with and without smearing; in single and double precision; on both P100 and TITAN V.

The absolute timings for the best performing implementations for all four cases, in both single and double precision on both GPUs, are shown in Figure 7. There are two graphs for each GPU used in the measurements. The execution time for single precision is shown on the left, and for double precision on the right. The times for the original version are plotted in grey.

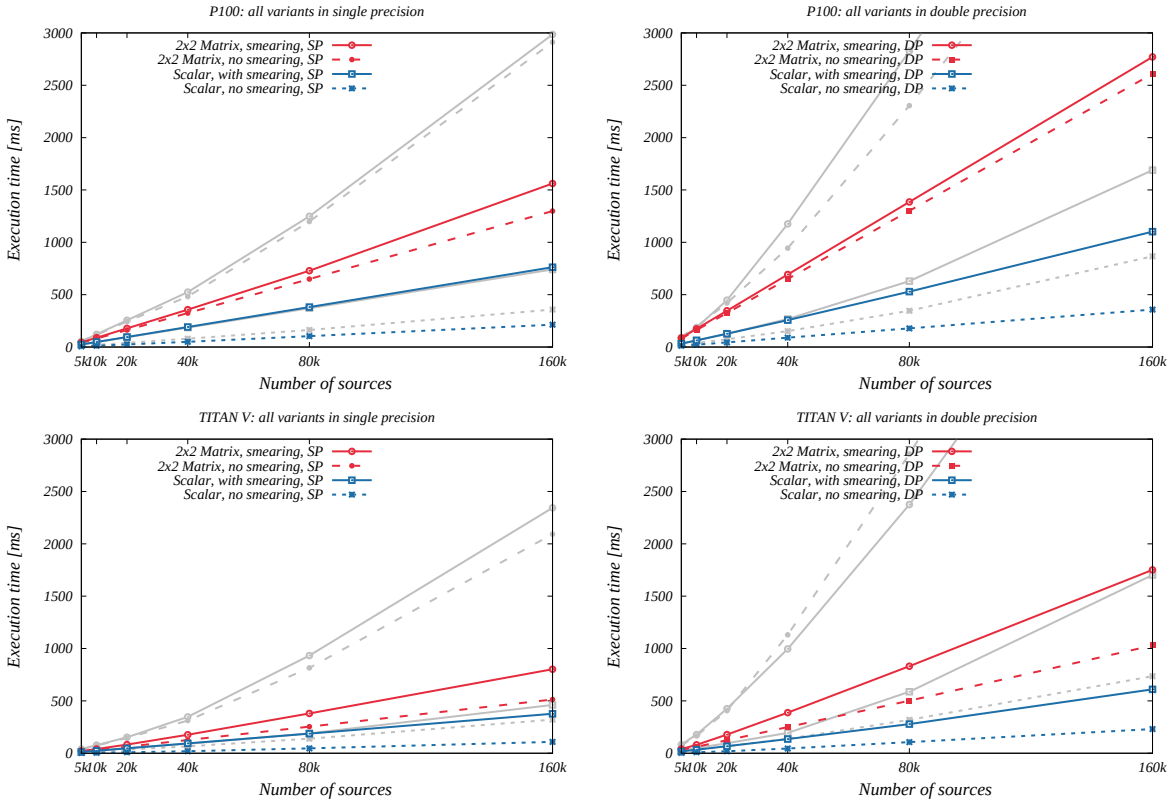

Figure 7: Execution time for P100 (top) and TITAN V (bottom) for all four cases of the predict algorithm we investigated. The execution time for single precision is to the left and execution time for double precision is on the right. The execution time of the original code is in grey. All measurements are for 512 stations, or 130,816 baselines.

The performance in FLOPs is shown in Figure 8. The figure shows the results with smearing in solid lines, and without smearing in dashed lines. Again, results from the P100 are shown at the top, TITAN V at the bottom; single precision is to the left and double precision to the right. We see that in some cases the performance declines with increasing number of sources. There are also cases where performance remains stable despite increasing number of sources.

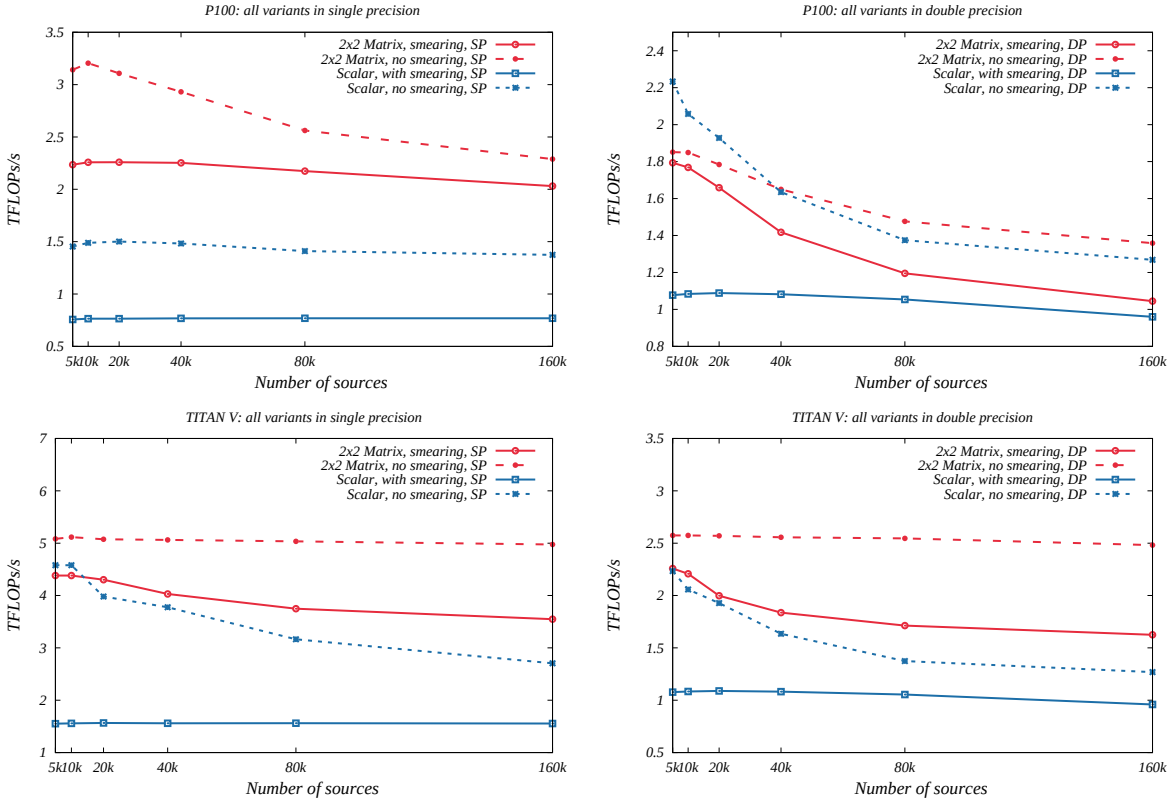

Figure 8: Number of floating point operations per second for P100 (top) and TITAN V (bottom) for all four investigated cases of the predict algorithm. The number of FLOPs/s for single precision is to the left, and double precision is to the right. All measurements are for 512 stations. These values are calculated from execution time and number of operations per source.

3.2.1 P100

The P100 GPU card is from the Pascal generation. The measured speed-ups in comparison with the original OSKAR code are shown in Figure 9 for predict with Jones matrices and in Figure 10 for predict with scalars.

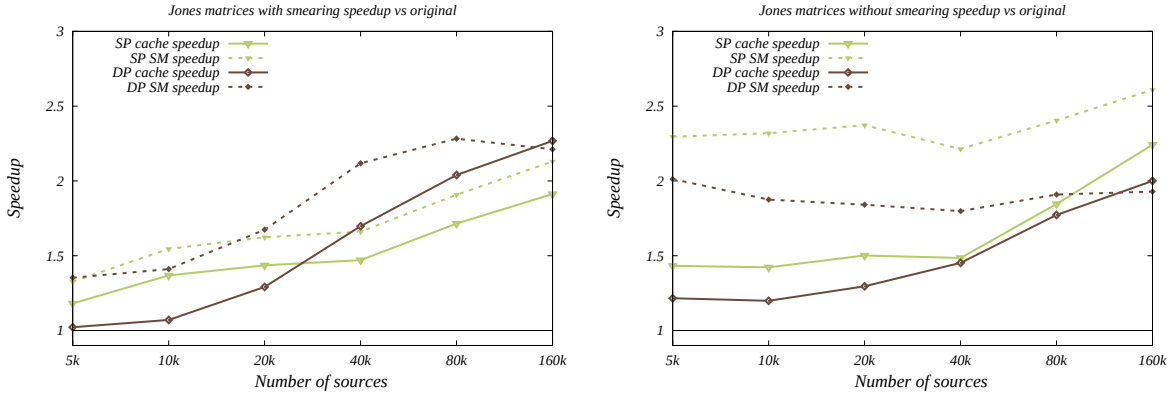

Figure 9: Speedup of the two optimised versions of the predict code, using cache or shared memory (SM), in comparison with the original OSKAR code on P100. The predict code uses Jones matrices, with smearing on the left side, without on the right. The results shown are for single precision (SP) and double precision (DP).

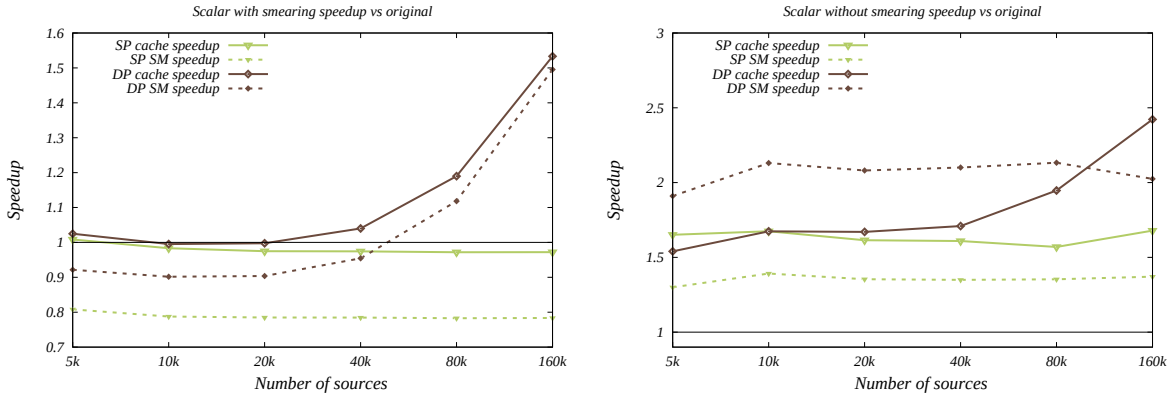

Figure 10: Speedup of the two optimised versions of the predict code, using cache or shared memory (SM), in comparison with the original OSKAR code on P100 producing scalar visibilities. The results shown are for single precision (SP) and double precision (DP).

3.2.2 TITAN V

The TITAN V GPU card is from the newer Volta generation. The measured speed-ups in comparison with the original OSKAR predict code are shown in Figure 11 for predict with Jones matrices and in Figure 12 for predict with scalars.

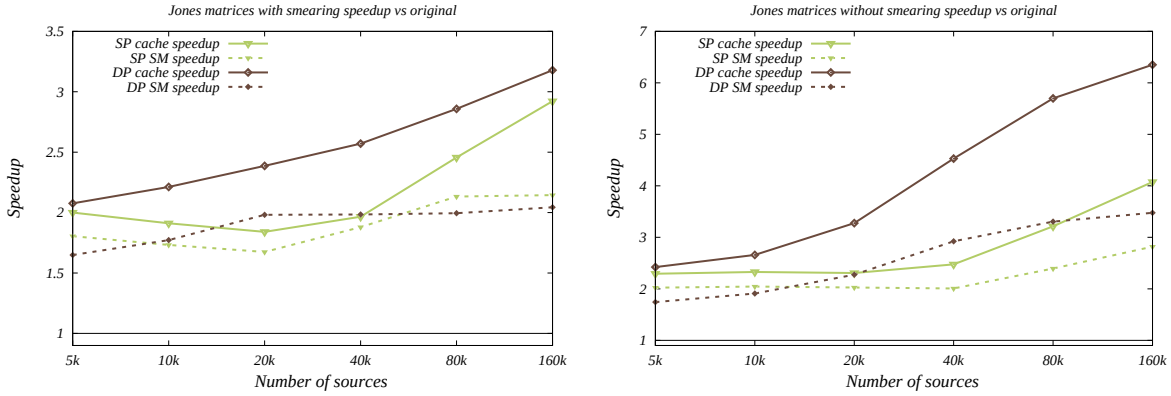

Figure 11: Speedup of the two optimised versions of the predict code, the cache version and the shared memory (SM), in comparison with the original OSKAR code. This is for the predict code which uses Jones matrices, with or without smearing on TITAN V. The results shown are for single precision (SP) and double precision (DP).

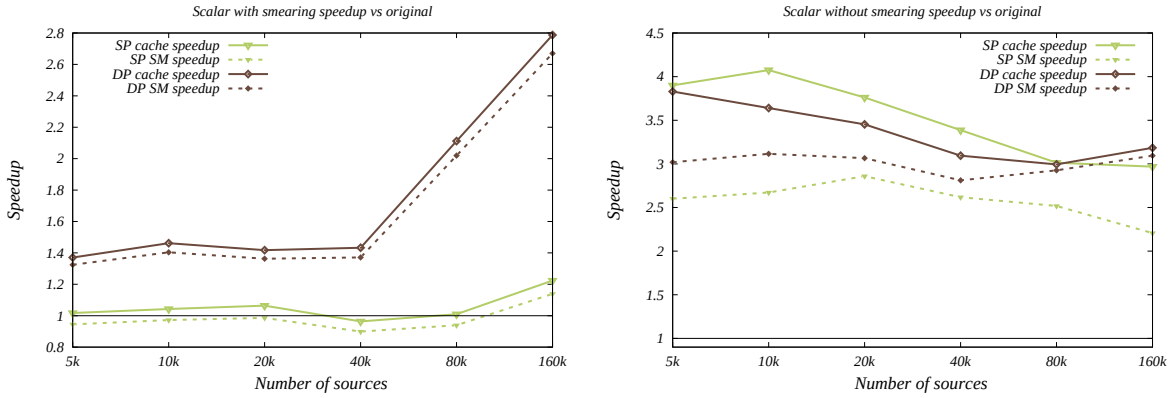

Figure 12: Speedup of the two optimised versions of the predict code, using cache or shared memory (SM), in comparison with the original OSKAR code on TITAN V producing scalar visibilities. The results shown are for single precision (SP) and double precision (DP).

3.3 Effect of optimisations on overall runtime of OSKAR

The effect of these optimisations on the overall run time of OSKAR depends on various simulation parameters. The aggregated speed-up for the OSKAR interferometer simulation is shown in Figure 13. The number of calculated visibilities per second is shown in Figure 14.

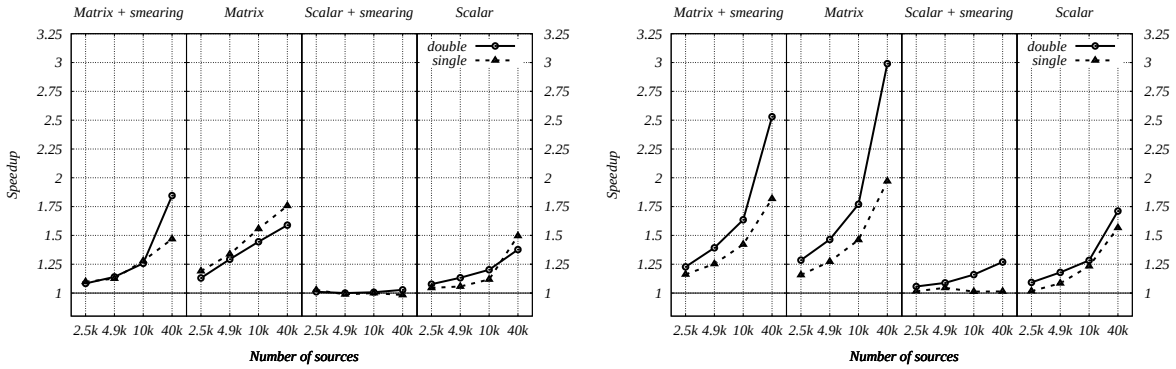

Figure 13: Speedup of the interferometer simulation in OSKAR after introducing optimised versions of the predict code (cache, shared memory) in comparison with the original performance on P100 (left) and TITAN V (right).

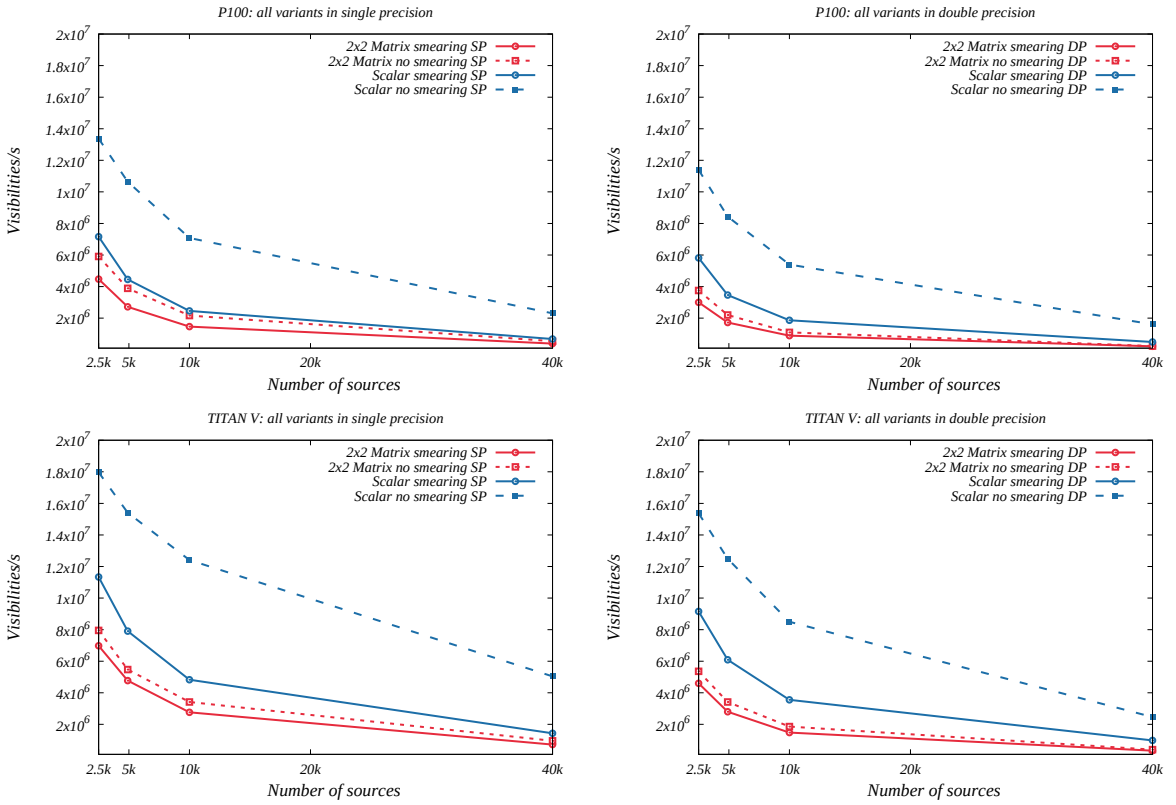

Figure 14: Number of calculated visibilities per second for P100 (top) and TITAN V (bottom) for all four investigated cases of the predict algorithm. The results for the single precision (SP) is on the left and for double precision (DP) on the right. These values are based on the execution time of simulations which use only simple Gaussian station beam evaluations. All results are for 512 stations, or 130,816 baselines.

4 Discussion

We first discuss execution time measurements shown in Figure 7. Looking at the P100 execution time for the matrix version of the predict code in double precision, we see that the non-smeared case has almost the same performance as with smearing. This is because both versions are limited by L2 cache bandwidth, so the predict code without smearing cannot perform any better then the version with smearing. On the P100 still, we see that the performance gain, when compared to the original code for the predict in single precision with smearing, is negligible or marginal. This is because the original predict code in OSKAR was already compute bound by the special function units. Hence our optimisations, which focused on data re-use, had no or negative effect. We attribute the lower performance of the optimised code in this case to extra synchronizations we introduced into the kernel. We see similar behaviour for TITAN V in single precision, although not in double precision. The most probable cause of this is poor caching properties of the original code, a behaviour which is not as apparent on the Pascal architecture. In general the performance of the optimised predict code in single precision is twice as fast as the version using double precision, which is expected. For the fully polarised version, the TITAN V GPU is about $1.60\times$ faster than the P100, and about $2\times$ faster in the scalar case.

In terms of number of floating point operations per second (Figure 8), we see that in some cases the number of FLOPs/s of the optimised predict code is constant or changing very little with source count. This indicates that the implementation scales well with problem size, and the efficiency of the code is the same (the number of cache misses does not increase; data are kept in caches). Constant performance of the code might also indicate that the code is limited by some other bottleneck. We see this in the scalar case with smearing, where performance is limited by the special function units used to evaluate the $\mathrm{sinc}()$ functions for the smearing terms. We also see that in some cases, performance of the optimised versions decreases with increasing number of sources. This indicates that the caching efficiency decreases as the number of sources grows larger, as demonstrated for example in the version of predict using Jones matrices with smearing in double precision. This happens when there are many thread blocks accessing different sources: data cannot then be shared between thread blocks, and must be read again from device memory. Splitting up a large source catalogue into smaller sections and accumulating these partial sums to form the total visibility might help to reduce these inefficiencies.

4.1 P100

The P100 is a scientific GPU card from the Pascal generation. From Figure 9 and Figure 10 we see that we achieved speed-up in all but two cases, where the predict operation is using complex scalars with smearing enabled. Performance improvements were not possible here because the original OSKAR code was already compute bound by the special function units. These are used for calculation of the $\mathrm{sinc}()$ functions in the smearing terms. The optimisations we introduced were targeted at enabling more data re-use, and so were ineffective in this case. However, in all other cases we achieved good speed-ups. In the case where Jones matrices are used without smearing we achieved speed-ups $1.8\times$ or more in double precision, and speed-ups $2.2\times$ or more in single precision.

4.2 TITAN V

The TITAN V is a high-end consumer GPU card from the Volta generation. The speed-up over the original OSKAR code is shown for the matrix version in Figure 11 and for the scalar version in Figure 12. Using the TITAN V, we achieved better speed-ups than we did for the P100. We achieved very good speed-ups for double precision in general: for the matrix versions (Figure 11), the speed-up is greater than $2\times$ even for low source counts, and the best achieved speed-up is more then $6\times$ when smearing is disabled. Similarly we achieved very good speed-up, in both single and double precision, for the case of the scalar version (Figure 12) without smearing, where the speed-up is more then $3\times$. Good speed-ups were also achieved for the matrix versions in single precision. The most problematic was again the scalar version with smearing enabled. The reason for poor performance here is the same as for the P100: the original OSKAR code was compute bound, and our work to optimise data re-use cannot change that.

5 Conclusion

For the cases where performance of the original version was limited by memory bandwidth, we improved the performance of the direct visibility prediction method described in this report over the original implementation by at least 30% when using the P100 GPU, and by at least 90% using the TITAN V GPU. The significant speed-ups in these cases were achieved by making better use of fast local memory on both GPUs to improve the level of data re-use, thus reducing the overall global memory bandwidth requirements. Because of architectural differences between the two generations of hardware, the best optimisation strategy depended on the type of GPU being used: the P100 GPU generally performed better by explicitly using on-chip shared memory as a user-managed cache, while the TITAN V GPU generally performed better by making better use of the dedicated cache. The performance gain also increased with increasing source count. For a small number of sources (below ~ 20 k), performance is limited by the special function units, which are used to evaluate $\mathrm{sinc}()$ functions for the smearing terms. The optimisations had no effect in the case where the original version was already limited by the compute performance of the special function units.

We also demonstrated that the optimisations introduced had a meaningful impact on the overall run-time of the OSKAR interferometer simulator. The most significant speed-up was obtained on the TITAN V GPU, where for the double precision matrix version we achieved $2.5\times$ and $3\times$ speed-up, respectively, with and without smearing enabled.

A How does the 'predict' stage in OSKAR differ to that in SKA pipelines?

OSKAR is an interferometer simulator, designed to generate simulated visibilities from radio telescopes containing aperture arrays, such as the SKA. It is freely available at http://github.com/OxfordSKA/OSKAR.

In OSKAR, visibilities for each source s are formed on baselines between pairs of stations p and q by evaluating the radio interferometer measurement equation (Hamaker et al., 1996), optionally using Jones matrices to handle polarisation (e.g. Smirnov, 2011). If full polarisation products are not required, the 2×2 complex Jones matrices $\mathbf{J}$ can be substituted by complex scalars to generate Stokes-I visibilities directly. However, in the rest of this section, full polarisation is assumed.

First, Jones matrices are generated for every visible source at every station in the interferometer. As discussed in Smirnov (2011), the Jones matrix formalism can model many physical effects, but the only really essential term to include for interferometry is the phase difference between station pairs. This is a scalar term, and is evaluated on a per-station basis with respect to the telescope coordinate origin. For a source with direction cosines (l,m,n) with respect to the phase tracking centre, the phase term $\mathbf{J}_k(p,s)$ at a station with (u,v,w) coordinates[3] is given by:

$$\mathbf{J}_k(p,s) = \exp\left\{ i\frac{2\pi}{\lambda}\left[u \cdot l + v \cdot m + w \cdot (n-1)\right]\right\} \tag{3}$$

The source brightness matrix $\mathbf{B}(s)$ is assembled from the source Stokes parameters (I,Q,U,V). In a linear polarisation basis, it is written:

$$\mathbf{B}(s) = \begin{bmatrix} I+Q & U+iV \\ U-iV & I-Q \end{bmatrix} \tag{4}$$

Pairs of Jones matrices are then multiplied together with the brightness matrix to evaluate the source visibility on each baseline. In a real instrument, the visibility amplitude will be reduced due to time-average smearing $\zeta_\tau(p,q,s)$ and bandwidth smearing $\zeta_\nu(p,q,s)$. If the source is extended, it can be modelled in OSKAR as a 2D Gaussian, so this is another per-source and per-baseline term $G(p,q,s)$. For a sky model containing multiple sources, these products are simply accumulated on-the-fly on each baseline as follows:

$$\boxed{\mathbf{V}(p,q) = \sum_s \zeta_\tau(p,q,s) \cdot \zeta_\nu(p,q,s) \cdot G(p,q,s) \cdot \mathbf{J}(p,s) \cdot \mathbf{B}(s) \cdot \mathbf{J}^H(q,s)} \tag{5}$$

Bandwidth smearing is modelled, assuming a square bandpass, as a sinc-function, where $\mathrm{sinc}(x) = \sin(x)/x$. In the following, (uu,vv,ww) are now the baseline vector coordinates in the (u,v,w) coordinate system, and $\Delta\nu$ is the bandwidth. The bandwidth smearing at each source on each baseline is given by (Taylor et al., 1999):

$$\zeta_\nu(p,q,s) = \mathrm{sinc}\left\{ \frac{\pi \cdot \Delta\nu}{c}\left[uu \cdot l + vv \cdot m + ww \cdot (n-1)\right]\right\} \tag{6}$$

Equation 6 shows that bandwidth smearing is most significant on long baselines for sources far from the phase centre.

The time-average smearing is modelled, to a very high approximation, as another sinc-function. During an integration time τ, the Earth rotates by an angle $\delta\theta = \omega_e\tau$, and any given

[3] The (u,v,w) and (l,m,n) coordinate systems are defined in Thompson et al. (2001).

baseline will move by an amount $(\delta u, \delta v, \delta w)$. These values are calculated using baseline components (xx, yy, zz) in the Earth-centred-Earth-fixed (ECEF) coordinate frame, and the Greenwich Hour Angle H_0 and Declination δ_0 of the phase tracking centre:

$$\begin{pmatrix} \delta_u \\ \delta_v \\ \delta_w \end{pmatrix} = \delta\theta \begin{pmatrix} xx \cdot \cos H_0 - yy \cdot \sin H_0 \\ (xx \cdot \sin H_0 + yy \cdot \cos H_0) \cdot \sin \delta_0 \\ -(xx \cdot \sin H_0 + yy \cdot \cos H_0) \cdot \cos \delta_0 \end{pmatrix} \tag{7}$$

The time-average smearing at each source on each baseline is given by (Taylor et al., 1999):

$$\zeta_\tau(p,q,s) = \mathrm{sinc}\left\{ \frac{\pi}{\lambda} \left[\delta u \cdot l + \delta v \cdot m + \delta w \cdot (n-1) \right] \right\} \tag{8}$$

Equation 8 shows that time smearing is most significant on baselines that move rapidly, at short wavelengths, for sources far from the phase centre.

Since OSKAR was designed to simulate visibilities from sources anywhere in the sky, the smearing terms are evaluated as described above if time and bandwidth smearing are enabled. However, to improve performance and reduce the required memory bandwidth, simpler kernels are used if Stokes-I-only ('scalar') mode is selected.

Visibility prediction for SKA pipelines

A general algorithm for visibility prediction is hard to define, as prediction covers all cases where simulated visibilities must be produced. In the context of an SKA pipeline, a prediction stage is likely to be used in at least two places:

1. To produce model visibilities to calibrate against.
2. To produce model visibilities from a clean-component list, as part of the image deconvolution process if subtraction is performed in the visibility domain.

Both stages are likely to have somewhat different requirements.

When generating model visibilities for calibration, it will be important to model all known effects that cannot be calibrated out – if these effects are not modelled properly, they will introduce errors into the calibration. If the model includes bright sources far from the phase tracking centre, then it may be necessary to model the effects of time smearing and/or bandwidth smearing on these sources.

When generating model visibilities for subtraction as part of image deconvolution, smearing effects are anticipated to be much less significant, since sources within the field are relatively close to the phase centre. The deconvolution process may therefore not need to model the effects of time smearing or bandwidth smearing, and this would be a considerable computational saving.

Also, if the sky model consists of an image at the phase centre instead of randomly scattered sources, and if smearing effects can be neglected, it may instead be much more efficient for a predict stage to take the fast Fourier transform (FFT) of the sky image, and then de-grid the transformed data onto the projected baseline coordinates. This approach avoids the need to compute the explicit sum over pixels (or sources) as in Equation 5, and the phases are generated by the FFT instead of by direct evaluation as in Equation 3; however, it is also a less flexible approach.

Depending on the requirements of the algorithms used for calibration and deconvolution, it it possible that neither will need to use the full Jones matrix formalism in their predict

stages – it will probably be sufficient to use complex scalars instead of 2×2 complex matrices. However, it may still be beneficial to evaluate station- rather than baseline-based terms where this is possible, since for N stations, the number of baselines will be a factor $(N-1)/2$ larger. For SKA1-LOW where $N = 512$, the station-based approach would reduce by a factor of ~ 256 times the number of sine and cosine computations required to evaluate the phase terms.

References

Hamaker, J. P., Bregman, J. D., and Sault, R. J.: Understanding radio polarimetry. I. Mathematical foundations., A&AS, 117, 137–147, 1996.

Smirnov, O. M.: Revisiting the radio interferometer measurement equation. I. A full-sky Jones formalism, A&A, 527, A106, 2011.

Taylor, G. B., Carilli, C. L., and Perley, R. A., eds.: Synthesis Imaging in Radio Astronomy II, vol. 180 of *Astronomical Society of the Pacific Conference Series*, 1999.

Thompson, A. R., Moran, J. M., and Swenson, Jr., G. W.: Interferometry and Synthesis in Radio Astronomy, 2nd Edition, 2001.

Williams, S., Waterman, A., and Patterson, D.: Roofline: An Insightful Visual Performance Model for Multicore Architectures, Commun. ACM, 52, 65–76, http://doi.acm.org/10.1145/1498765.1498785, 2009.