SDP Memo 089: SKA-SDP Reprojection on Graphical Processing Units

Document number……SDP Memo 089
Document Type……MEMO
Revision……...1
Author……..NVIDIA Corporation
Release Date……...2018-10-25
Document Classification……..…… Unrestricted

Lead Author:
NVIDIA Corporation

Released by:

Name	Designation	Affiliation
Bojan Nikolic	SDP Project Engineer	University of Cambridge
Signature & Date:		

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.

SKA-SDP Reprojection on Graphical Processing Units

NVIDIA Corporation, 2701 San Tomas Expressway,
Santa Clara, CA 95050 USA

1. SKA Reprojection

1.1. Basic Algorithm

In radio astronomy, reprojection is the mapping of image data from one plane or other projection onto another. The images produced by the SKAs low- and mid-range arrays are in a plane tangential to the Earth's surface. But, as the Earth rotates, this plane moves. In order to sample over a longer period, it is helpful to be able to project images from different planes onto a single plane (see Figure 1). To do so, we first project the image onto a world coordinate system expressed in terms of latitude and longitude. Then, we project from that coordinate system onto the final tangential plane to get a final image.

In practice, we create a grid of points in the final plane, transform those to the initial plane, then interpolate those points in the original image to get a new, projected image.

1.2. Projection

For SKA, the appropriate projection is an oblique parallel projection. We took as our guide the SIN projection from wcslib developed by Mark Calabretta [1].

1.3. Interpolation

Once the points from the final grid are projected onto the original grid, we interpolate to find the values from the original image at these new points. By default, we use a bicubic b-spline interpolation of degree three. This method

[1] http://www.atnf.csiro.au/people/mcalabre/WCS/wcslib/

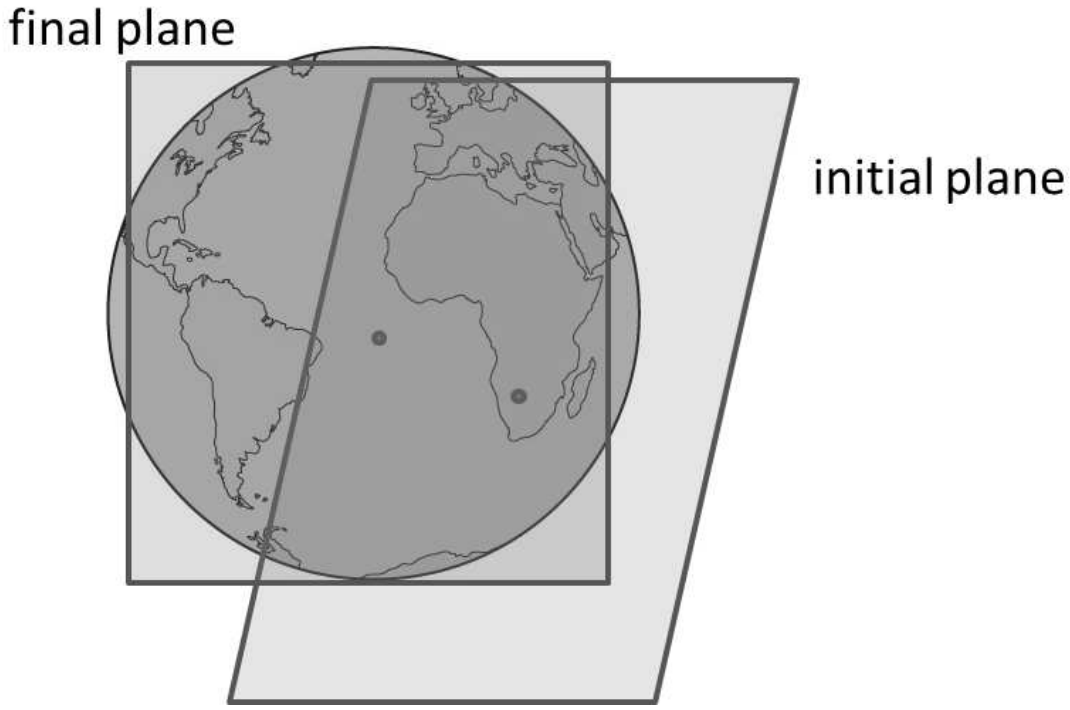

Figure 1: Reprojection between tangential planes

constructs the new value from image points up to 2 grid points away. For a one-dimensional interpolation the new value can be expressed as

$$f'(x_0+\alpha) = w_0(\alpha)f(x_0-1)+w_1(\alpha)f(x_0)+w_2(\alpha)f(x_0+1)+w_3(\alpha)f(x_0+2) \quad (1)$$

where $f(x)$ is the original function, x_0 is strictly on-grid in the original image and α is less than one and greater than or equal to zero. The coefficients

w_i are given by

$$w_0(\alpha) = \frac{1}{6}(-\alpha^3 + 3\alpha^2 - 3\alpha + 1) \quad (2)$$

$$w_1(\alpha) = \frac{1}{6}(3\alpha^3 - 6\alpha^2 + 4) \quad (3)$$

$$w_2(\alpha) = \frac{1}{6}(-3\alpha^3 + 3\alpha^2 + 3\alpha + 1) \quad (4)$$

$$w_3(\alpha) = \frac{1}{6}\alpha^3 \quad (5)$$

An interpolation on the 2-dimensional image can be accomplished by interpolating in each direction separately. That is, to interpolate for a point $x_0 + \alpha, y_0 + \beta$ within one grid point of, the on-grid point x_0, y_0, we separately interpolate the points $(x_0 + \alpha, y_0 - 1)$, $(x_0 + \alpha, y_0)$, $(x_0 + \alpha, y_0 + 1)$ and $(x_0 + \alpha, y_0 + 2)$. Then, we use those four points to interpolate for $(x_0 + \alpha, y_0 + \beta)$.

2. GPU implementation

The projection step is a simple appropriation of the SIN projection from the wcslib. CUDA supports all the necessary trigonometric functions. As will be discussed below, certain trigonometric functions have explicit fast implementation in the GPU hardware for exceptional performance. The interpolation step could, additionally, be simply implemented on the GPU. Image interpolation is a critical function for the GPU and we can exploit the GPU's native interpolation functions to improve performance here.

2.1. Use of texture in interpolation

Textures on NVIDIA GPUs implement a linear interpolation in the hardware. A texture read will efficiently return

$$f(x_0 + \alpha) = (1 - \alpha)f(x_0) + \alpha f(x_0 + 1)$$

As explained in GPU Gems[2], any linear combination of $f(x_0)$ and $f(x_0 + 1)$, e.g.

$$af(x_0) + bf(x_0 + 1)$$

[2] *GPU Gems 2: programming techniques for high-performance graphics and general-purpose computation*. Ed. Matt Pharr. Addison-Wesley (2005).

can be expressed as

$$(a+b)f\left(x_0+\frac{b}{b+a}\right)$$

with the value of f computed via a texture read. An additional requirement for this computation is that $b/(b+a)$ be between zero and one (inclusive).

We can use this to speed up the computation of equation (1).

$$w_0 f(x_0-1)+ \quad w_1 f(x_0)+w_2 f(x_0+1)+w_3 f(x_0+2) = \tag{6}$$

$$(w_0+w_1)f\left(x_0-1+\tfrac{w_1}{w_0+w_1}\right)+ \tag{7}$$

$$(w_2+w_3)f\left(x_0+1+\tfrac{w_2}{w_2+w_3}\right) \tag{8}$$

Here, we replaced four on-grid texture accesses with two off grid texture accesses, reducing the number of texture instructions and utilizing the texture unit for some of the floating point operations.

One important downside to this implementation is that hardware interpolation for textures is not available for double precision. For double precision, textures can still be read for on-grid values, but since they are read as int2 values then cast to a double, interpolation is not possible.

3. Performance

3.1. Single precision vs. double precision

GPUs provide several hardware specialisations that are ideal for reprojection. Unfortunately, these are available only for single precision. In addition, these hardware tricks may change results.

The table below gives performance numbers and the largest error for different parameters when compared with a double precision implementation on a CPU. This test case uses a random image to produce an outside estimate of the error. The error recorded is the maximum discrepancy between a double precision CPU computation and the specified GPU computation. The image consisted of random values between 0 and 1. Performance is expressed in billions of points reprojected per second. The image was 4096x4096 and complex-valued. The hardware used is a K40c with clocks boosted to 875 MHz and ECC disabled. Data transfer time to the GPU was not included.

In the table, we have separated the scheme and computational precision for reprojection of the grid points and interpolation on the original image.

Reprojection can be either double or single precision and with and without fastmath, a set of speedups to trigonometric functions, square root and inversion. These can introduce additional error as seen below.

Interpolation can use either single or double precision and with or without reading through texture memory. For double precision, we are restricted to an on-grid texture read, followed by 16 fused multiply-add (FMA) operations. For single precision, we have the option to utilize the fast texture interpolation provided on in the GPU hardware as discussed in section 2.1. The code used here can be found in the SDP github repository[3].

Projection Scheme	Interpolation Scheme	Gpixels/sec	Maximum error
DP, no fastmath	DP, no texture	0.21	0
DP, no fastmath	DP, on-grid texture	0.81	0
DP, fastmath	DP, on-grid texture	0.89	0.0000002
SP, no fastmath	DP, on-grid texture	0.82	0.0011
SP, fastmath	DP, on-grid texture	0.89	0.4
DP, fastmath	SP, on-grid texture	1.50	0.000087
DP, fastmath	SP, texture interpolation	1.62	0 .0017
SP, no fastmath	SP, on-grid texture	1.96	0.0011
SP, no fastmath	SP, texture interpolation	2.28	0.0021
SP, fastmath	SP, texture interpolation	3.16	0.4

Table 1: Performance and maximum error for various reprojection schemes on the GPU

The error for the single precision implementation with fastmath warrants some explanation. In this case, approximations in the math functions, specifically sqrt(), moved the reprojected location by just 0.0008 grid points, but this was enough to move the point from just above a grid point, to just below the same point. In the random image, this change presents a significantly different set of points to the FIR filter, giving is a starkly different result. A real image is expected to be much smoother and less prone to significant errors of this type.

[3]https://github.com/SKA-ScienceDataProcessor/GPUReprojection

3.2. Dependence on image size

Since each image point is mapped to a single GPU thread, smaller images can reduce the amount of parallelism exposed to the GPU and "starve" the cores. Table 3.2 shows performance for a range of image sizes. Here, we use double precision reprojection with fastmath and single precision interpolation with on-grid texture. The hardware, as before, is a K40c with boosted clocks.

Image dimension	Performance (Gpixels/sec)
8192	1.67
4096	1.61
3072	1.49
2816	1.35
2048	0.95
1024	0.61

Table 2: Performance for varying image size

As shown in the table and figure 2, performance begins to drop for image sizes less than 4096x4096.

3.3. Comparison with FFT

Reprojection as implemented dispatches 156 million warp-level instructions to reproject a 8192x8192 complex image. This is about 312 floating point operations per grid point. The computation time for this is 98.1 ms using only double precision. A complex-to-complex FFT on the same size image should be $5 * 8192 * 8192 * \log(8192)/32 \simeq 130$ million warp-level operations. This operation takes 44.2 ms. Reprojection makes slightly better use of the GPU ($312/98.1 > 130/44.2$) due to the very local nature of the projection operation. The FFT, by contrast, must interleave computation with data fetches, reducing performance. The scaling of these two operations is, of course, different. Reprojection scales as $\mathcal{O}(n^2)$ while FFT scales as $\mathcal{O}(n^2\log(n))$. So, for smaller image sizes, reprojection will be even most costly relative to FFT.

3.4. Maxwell performance

NVIDIA has not released a Tesla product based on the new Maxwell architecture. As such, we cannot do a direct comparison of double precision

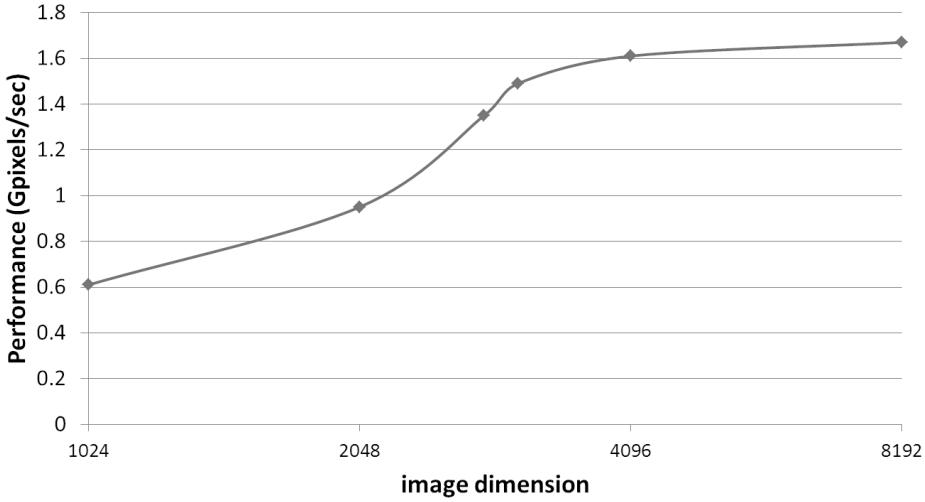

Figure 2: Reprojection performance for varying image size

performance between Kepler and Maxwell. We can do a comparison for single precision computation. We repeated the computation from table 3.1 on a Quadro M6000 using single precision for both steps, no fastmath and texture interpolation. For the K40c, from the GK110 Kepler architecture, the throughput was 2.28 Gpixels/s. For the Quadro M6000, based on the GM200 Maxwell architecture, throughput was 3.36 Gpixels/s. This improvement is slightly better than we would expect from the increase in the clock frequency and the number of cores. The K40 has 2880 and operates (boosted) at 875 MHz. The Quadro M6000 has 3072 cores at 1151 MHz for an expected performance boost of 1.40. The observed improvement is 1.47.

3.5. Extrapolation to Pascal

Reprojection on the GPU is strongly limited by computational throughput, specifically the performance of the fused multiply-add (FMA) operation. When single precision is used for interpolation, the computation is nicely bal-

anced between single and double precision operations. We therefore expect performance to scale with the product of the clock speed and the number of cores. For Pascal, this is expected to be 4X higher than for Kepler.

3.6. Code repository

The code used for all of the above work can be found in the git repository located at https://github.com/SKA-ScienceDataProcessor/GPUReprojection. The README file found there explains how to build and run the code.

4. Conclusions

In this report, we have explored performance of reprojection on the GPU and report the following findings.

- Reprojection takes slightly more than twice as much time as a complex-to-complex FFT for a 8192x8192 complex image. This is excellent utilization of the GPU.
- Computation time can be reduced by utilizing fastmath operations for some math operations and by utilizing accelerated interpolation provided by GPU textures. Both of these increase the error in the calculation.
- Scaling to the Maxwell architecture is slightly better than expected. The upcoming Pascal architecture is expected to provide a 4X improvement over the numbers reported here.

Further work could include investigating the effects of individual fastmath operations and exploring the power consumption of the GPU during reprojection.