A Study on Convolution Operator Using Half Precision Floating Point Numbers on GPU for Radioastronomy Deconvolution

Mickael Seznec, Nicolas Gac, André Ferrari, François Orieux

A Study on Convolution Using Half-Precision Floating-Point Numbers on GPU for Radio Astronomy Deconvolution

Mickaël Seznec[1], Nicolas Gac[1], André Ferrari[2], François Orieux[1].

[1] Laboratoire des Signaux et Systèmes (L2S), CentraleSupélec, CNRS, Univ Paris sud, Université Paris Saclay, Gif-sur-Yvette, France
[2] Lab. J.-L. Lagrange, Université de Nice Sophia Antipolis, CNRS, Observatoire de la Côte d'Azur, Parc Valrose, F-06108 Nice cedex 02, France

***Abstract*— The use of IEEE 754-2008 half-precision floating-point numbers is an emerging trend in Graphical Processing Units' architecture. Being such a compact way of representing data, its use may speed up programs by reducing the memory bandwidth usage and allowing hardware designers to fit more computing units within the same die space. In this paper, we highlight the acceleration offered by the use of half floating-point numbers over different implementations of the same operation, a 2D convolution. We show that even though it may lead up to a significant speed-up, the degradation brought by this new format is not always negligible. Then, we choose a deconvolution problem inspired by the SKA radio-telescope processing pipeline to show how half floats behave in a more complex application.**

***Index Terms*— deconvolution, radio astronomy, half-precision floating-point, GPU, parallel computing**

I. Introduction

Before appearing in the IEEE standard in 2008 [1] as binary16, half-precision floating-point arithmetic (FP16) has been a topic of interest for computer graphics community since the early 2000s. In parallel, embedded high-performance computing [2] has also investigated its use as an alternative to fixed-point arithmetic in order to design more energy-efficient hardware accelerators. In the same way, deep learning has made a renewed interest to approximate computing [3], [4] especially since GPUs provide half float computation [5]. Indeed, NVIDIA GPUs are offering half float storage since 2015 with CUDA 7.5, half float Multiplier-ACcumulator (MAC) since 2016 with the Pascal architecture [6] and tensor cores, designed for convolutional neural network training, since 2017 with the Volta architecture [7]. These tensor cores offer a Fused-Multiply-Add (FMA) with a mixed precision: a half float multiplication of the FP16 operands followed but an accumulation in the FP32 format.

The half-precision floating point format occupies only 16 bits (1 bit of sign, 5 bits of exponent and 10 bits of mantissa) as illustrated on fig 1 whereas single precision occupies 32 bits (8 bits of exponent and 23 bits of mantissa). Compared to 16-bit integers, it offers an increased dynamic range and compared to 32-bit reals, it divides by 2 the memory storage and bandwidth, of course at the cost of a reduced precision and range. Moreover, the theoretical peak performance (Tflops) on NVIDIA GPU architectures can be significantly increased thanks to half-precision. For instance, the computation power of the Tesla V100 is 15.7 Tflops for FP32 MACs, 31.4 Tflops for FP16 MACs and 125 Tflops for tensors cores.

This work was supported by MAGELLAN (ANR-14-CE23-0004-01)

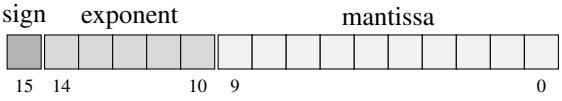


Fig. 1: IEEE 754 binary16 format

Acceleration and energy-efficiency brought by FP16 computation have to be put in the balance with the potential loss of precision. Stability of algorithms using FP16 format is an open problem [8]. Intuitively, one may think that applications where the raw data output of the instrument is integer values, with a dozen bits of accuracy, would not be too much penalized by this compressed-number representation. Like what has been observed for tomography reconstruction [9]. The goal of this study is to observe its use for another inverse problem, the deconvolution for radio astronomy. The optimization algorithm studied (gradient descent) is mainly based on the 2D convolution operator whose acceleration on GPU has been widely investigated for single-precision [10], [11], [12] but as far as we know not yet for half precision. The motivation of this paper is to benefit from the potential acceleration of the 2D convolution with FP16 on GPUs in the perspective of the SKA data processing challenge.

The remainder of this article is organized as follows. Section II describes the deconvolution problem solved using a simple gradient descent. Section III makes a benchmark of 2D convolution on GPU in terms of acceleration and precision. Section IV studies its application for image reconstruction in radio astronomy. Section V presents a discussion and an analysis of the experimental results.

II. Deconvolution

Deconvolution is a classical inverse problem [13] and arises when the observation model is a convolution

$$\boldsymbol{g} = \boldsymbol{H}\boldsymbol{f} + \boldsymbol{n} \tag{1}$$

where $\boldsymbol{g} \in \mathbb{R}^N$ is the data set, $\boldsymbol{f} \in \mathbb{R}^M$ the unknown, $\boldsymbol{n} \in \mathbb{R}^N$ unknown noise and $\boldsymbol{H} \in \mathbb{R}^{N \times M}$ the linear observation model or the convolution operator. If the convolution is circulant, then $N = M$, the matrix $\boldsymbol{H}$ is square and

diagonalizable in Fourier space like $\boldsymbol{H} = \boldsymbol{F}^\dagger \boldsymbol{\Lambda} \boldsymbol{F}$ where $\boldsymbol{F}$ is the linear Fourier transform and $\boldsymbol{\Lambda}$ a diagonal matrix. If the convolution is not circulant the matrix $\boldsymbol{H}$ is not necessary square but remains Tœplitz: all lines of $\boldsymbol{H}$ are shifted version of the first line. In both cases, the matrix $\boldsymbol{H}$ usually leads to ill-conditioned problems with instability and noise amplification.

A standard approach for the reconstruction relies on the regularized least-square where the solution $\hat{\boldsymbol{f}}$ is defined as the minimizer of a data adequacy term and a penalization term

$$\boldsymbol{J} = \|\boldsymbol{g} - \boldsymbol{H}\boldsymbol{f}\|^2 + \lambda \|\boldsymbol{f}\|^2 \tag{2}$$

$$\hat{\boldsymbol{f}} = \arg\min_{\boldsymbol{f}} \boldsymbol{J}(\boldsymbol{f}) \tag{3}$$

The penalization term $\|\boldsymbol{f}\|^2$ on the energy of the solution allows compensating the pathological behavior of the data adequacy and, depending on the balance term λ, leads to a well-posed problem with good conditioning.

The explicit minimizer is known

$$\hat{\boldsymbol{f}} = \left(\boldsymbol{H}^t\boldsymbol{H} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{H}^t \boldsymbol{g} \tag{4}$$

and is called the Wiener filter when $\boldsymbol{H}$ is diagonalizable in Fourier space. Otherwise, if the dimension of $\boldsymbol{f}$ is large, the size of the Hessian matrix $\boldsymbol{H}^t\boldsymbol{H}$ forbids the matrix inversion and the solution $\hat{\boldsymbol{f}}$ must be computed with an iterative linear solver [14]. A common one is the gradient descent or conjugate gradient descent described algorithm 1. We consider two versions: one with a fixed step α and one with the optimal one (that corresponds to the maximum descent in the current direction $\boldsymbol{r}$) that needs a little extra computation.

Algorithm 1 The gradient descent algorithm

Require: $\boldsymbol{H}$, λ, $\boldsymbol{g}$, ϵ, N, c
1: Set $\boldsymbol{b} = \boldsymbol{H}^t\boldsymbol{g}$ and $\boldsymbol{Q} = \boldsymbol{H}^t\boldsymbol{H} + \lambda\boldsymbol{I}$
2: Set $\boldsymbol{f}^{(0)}$ and $n \leftarrow 0$
3: **repeat**
4: $\quad \boldsymbol{k} \leftarrow \boldsymbol{H}^t\boldsymbol{H}\boldsymbol{f}^{(n)} - \boldsymbol{b} + \lambda\|\boldsymbol{f}^{(n)}\|^2$
5: $\quad \alpha \leftarrow \boldsymbol{k}^t\boldsymbol{k} / \boldsymbol{k}^t\boldsymbol{Q}\boldsymbol{k}$ ▷ or $\alpha \leftarrow c$ with $c \leq \frac{2}{\|\boldsymbol{Q}\|}$
6: $\quad \boldsymbol{f}^{(n+1)} \leftarrow \boldsymbol{f}^{(n)} - \alpha\boldsymbol{k}$
7: $\quad n \leftarrow n + 1$
8: **until** Some criterion is met ▷ e.g.: $n \geq N$
return $\boldsymbol{f}^{(n)}$

III. Convolution Benchmark

The algorithm shown in the previous section relies heavily on the convolution operator: two are needed to find k, and two supplementary ones for the optimal α. Computing a convolution is time-consuming and is often the bottleneck of such methods. There are many ways of implementing this operation on GPU [10]. In this section, we focus on the usage of half floating-point numbers for those methods.

Four implementations are compared in this benchmark: cuBLAS, cuDNN, cuFFT, naive and PRCF. The first three are part of libraries written by Nvidia. cuBLAS is an implementation of the BLAS API [15]. cuDNN (CUDA Deep Neural Network) is a low-level API for deep learning primitives used by other frameworks such as TensorFlow, Caffe2 or PyTorch [16]. cuDNN itself relies on different methods to perform a convolution, depending on many factors: the size of the convolution kernel, whether the images are batched [17]... cuFFT is a GPU implementation of the Fast Fourier Transform method to compute a discrete Fourier transform.

In addition to the implementations found in these libraries, we tested our own algorithms. "naive" launch one GPU thread per image pixel. It then loops over the convolution kernel to perform the multiply-add accumulation. A mixed strategy has also been tested: data are stored in half precision and computation are done using floats. In the kernel code, the GPU threads convert data to floats, do the computation in float and then convert the result back to half. Finally, we used "PRCF" (Parallel Register-only Convolution Filter), that was first presented by Perrot et al [11]. We re-implemented their method but instead on relying on a fixed code generator, we took advantage of C++ templates.

For this benchmark, we use a zero-padding method to handle border issues. Convolutions are done out-of-place. We first transfer the data to the GPU, time 20 convolutions to average the results, stop the timer and transfer the data back to the CPU to check the accuracy of the resulting image. The convolution kernel used is Gaussian and the image is a standard 512×512 pixels cameraman picture. The GPU used is a Nvidia Titan V [7].

In figure 2, the cuFFT curve is almost flat. In fact, the sum of the kernel's width and the image's width is padded to the next power of 2 for performance reasons. In this case, it's always 1024, hence these constant results. cuBLAS and cuDNN are way slower than our custom methods. In fact, cuDNN relies on a matrix multiplication method, just like cuBLAS. They are however useful in neural network contexts as they scale well when many kernels are to be convolved with the same image.

A gap in computation time appears in both "naive" and PRCF implementations for a kernel size of 35 or more. This is due to those implementations being loop unrolled for smaller kernels. However, the compilation time explodes as the size of kernel increase: after 35 we chose to tell the compiler not to optimize them. Finally, because we store the kernel in the GPU texture cache, those implementations are also limited by its size. Once it is too big to fit in, we cannot use them directly.

In terms of performance, once the kernel becomes big (between 35 and 50, depending on implementations and optimizations) it is faster to use a Fourier transform to compute the convolution. We will use this result when choosing an implementation in part IV.

When using half-precision floats, acceleration depends a lot on the chosen algorithm. In CUBLAS, it can reach a x4 speedup (see figure 3). This is due to the library using NVidia Tensor Cores to perform matrix multiplications[7]. The results for the remaining implementations are a bit

disappointing. For the naive algorithm, however, performance increases with a bigger kernel. In fact, the speedup is mainly explained by fewer memory transfers, that bound the algorithm when the kernel becomes large. Regarding PRCF, the poor performance might be due to worse compiler optimizations. Finally, in cuFFT, it is harder to give a justification as we do not have access to the code. Our explanation is that because memory issues do not coalesce, the bandwidth is not saturated, hence no real improvement when using FP16.

In figure 4, the Mean Relative Error (MRE) between the convolution computed on GPU and one done on CPU is displayed. MRE is computed as:

$$MRE = \frac{1}{N} \sum \begin{cases} \left| \frac{x[i] - y[i]}{x[i]} \right|, \text{if } x[i] \neq 0 \\ 0, \text{otherwise} \end{cases} \quad (5)$$

Where x and y are the images to compare and N the total number of pixels in an image.

Please note that we compared our own reference implementation with *convolve2d* from the Python package *SciPy*. The results are clear: when using half floats, the loss of precision is much higher. The error also increases with the size of the kernel. With a width of 115, cuFFT has a 1% error, naive and PRCF, a 10% error. This is due to the multiple imprecisions while accumulating the intermediate results. The naive mixed strategy (half storage and float computation) gives nearly the same acceleration than naive half but provides a lower error (10^{-4}) invariant to kernel size.

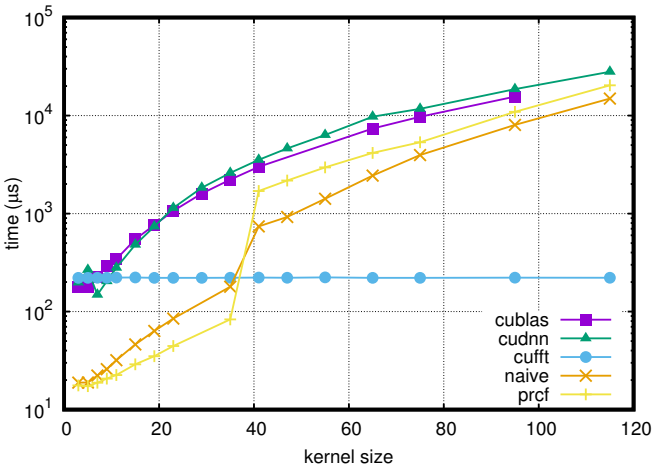


Fig. 2: Execution time in single precision

IV. Application to Image Reconstruction in Radio Astronomy

The future Square Kilometre Array (SKA) will provide radio interferometric data with unprecedented detail. To achieve the nominal performances of the instrument, image reconstruction algorithms are challenged to scale well with TeraByte image sizes never seen before. In the perspective of

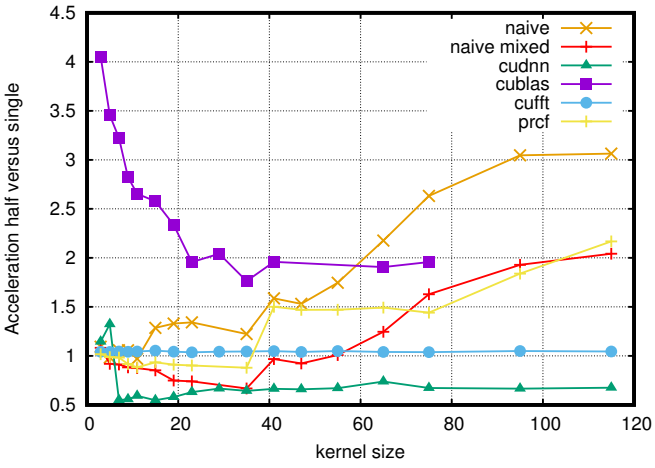


Fig. 3: Acceleration ratio in half vs single

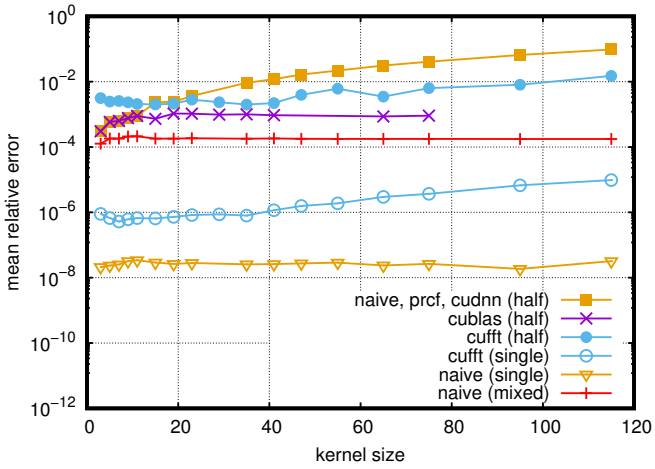


Fig. 4: Error compared to a reference implementation

this challenge, the simulations which follow focus on image deconvolution for radio astronomy.

We used a simulated PSF (Point Spread Function) for the SKA Phase 1 mid-frequency array. The array, which will include 197 dishes, will be built in South Africa from its precursor Meerkat. The PSF was obtained using the HI-inator package based on the MeqTrees software [18] (figure 5a). To ensure a high dynamic range, the simulated sky is composite of point sources and a faint halo modeled by a homogeneous Gaussian field (figure 5b). The ratio between the amplitude of the sources and the maximum value of the halo was set to 10^{-3}. The signal to noise ratio on the observed "dirty" image is set to 37dB.

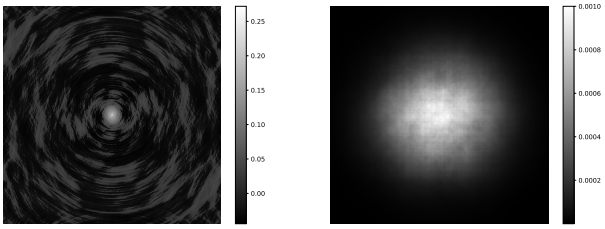
The goal is to reconstruct the image of the sky given a noisy and distorted observation. To accomplish that, we base our approach on the minimization of the quadratically penalized criterion (2) using a gradient descent algorithm as described in section II. Please note that the purpose of these simulations is not to illustrate the performances of the "state of the art" reconstruction algorithms but rather emphasizes advantages and shortcomings of using half-precision

floating-point numbers. All convolutions are done using cuFFT. On figure 6, you can observe multiple reconstructed images using different strategies and precisions. Criterion values across iterations are visible on figure 7. The balance term λ has been set to 0.01 as it provided sensible results.

The FP32 optimal-step curve represents the criterion value $\boldsymbol{J}$ across iterations of the algorithm described in section II (figure 7). As you can see, it quickly decreases and becomes almost flat. The same behavior is observed with the "float fixed-step" curve. In this method, the step α is constant. We chose it by looking at the optimal step values found in the first method and choosing the minimum one. The "mixed" curve behaves the same way. For this implementation, data is stored as halves but computations are made using floats.

The half-precision counterpart curves' behavior is slightly more complex. The optimal step method does not make the criterion decrease for every iteration, hence the noisy values. We can also notice a difference depending on the SNR (Signal-to-Noise Ratio): with a fixed step and a high (37dB) SNR, the criterion seems to decrease but only during the first 250 iterations. With more noise (16dB SNR) "half - fixed step" has the same behavior as the optimal step.

To address this issue, we try a different method: rather than blindingly using $\boldsymbol{f}^{(n+1)} \leftarrow \boldsymbol{f}^{(n)} - \alpha \boldsymbol{g}$, we use a backtracking algorithm. The criterion for the next iteration candidate is computed: if it is higher than the previous one, we step back, set $\alpha \leftarrow \frac{\alpha}{2}$ and try the new value. We proceed until the criterion decreases. Note that sometimes the computed gradient is so inaccurate that it is impossible to make the criterion decrease along its direction. When that happens (after a fixed number of retries), the procedure is ended. We then use the final image of this method as the starting point of an FP32 optimal step method. This is referred to as "half - backtracking than float -optimal step" in figure 7.

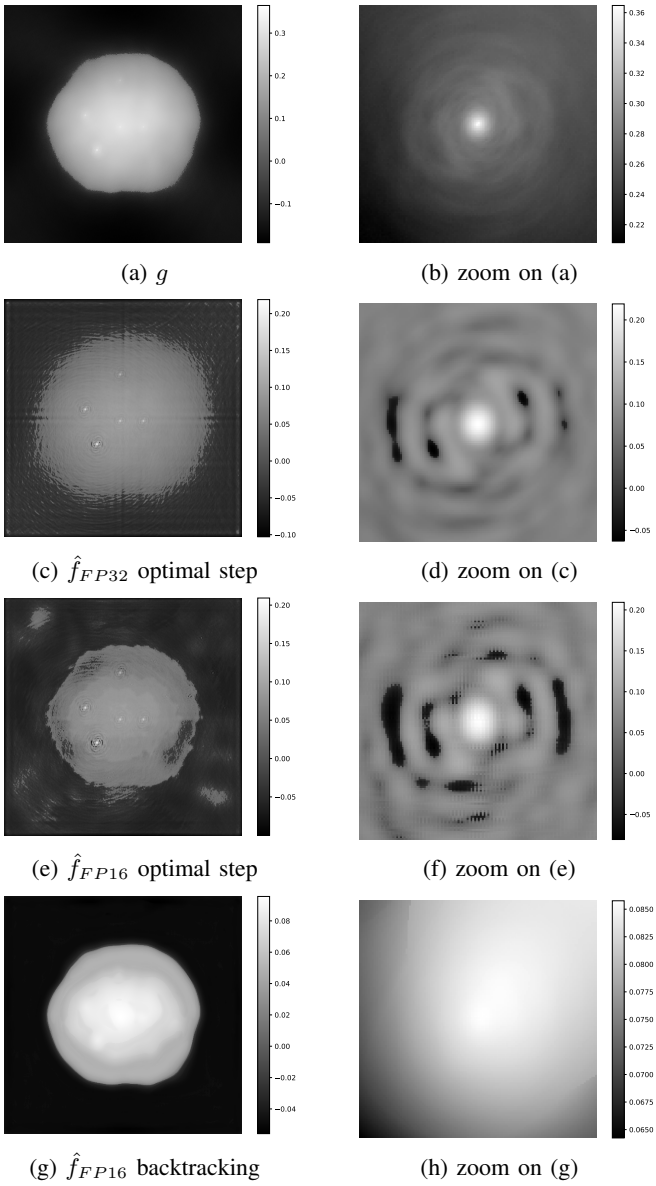


(a) PSF (cubic root) (b) Sky (clipped to 0.001)

Fig. 5: The dataset used

V. Discussion & Analysis

The first thing to point out is that it is difficult to rely on computations done using FP16 numbers. As seen in part II, when doing a convolution, the error increases with the kernel size. In part III, with a 2048x2048 kernel, it is not precise enough to make the criterion decrease at each step. On figure 8, the difference is striking across computations done with FP32 and FP16. The *MSE* between these two images is 9.46, with some points having a relative difference over 1000%.



(a) g (b) zoom on (a)

(c) $\hat{f}_{FP32}$ optimal step (d) zoom on (c)

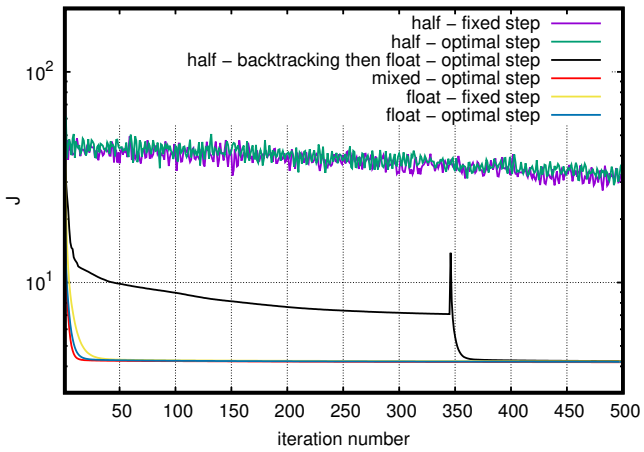
(e) $\hat{f}_{FP16}$ optimal step (f) zoom on (e)

(g) $\hat{f}_{FP16}$ backtracking (h) zoom on (g)

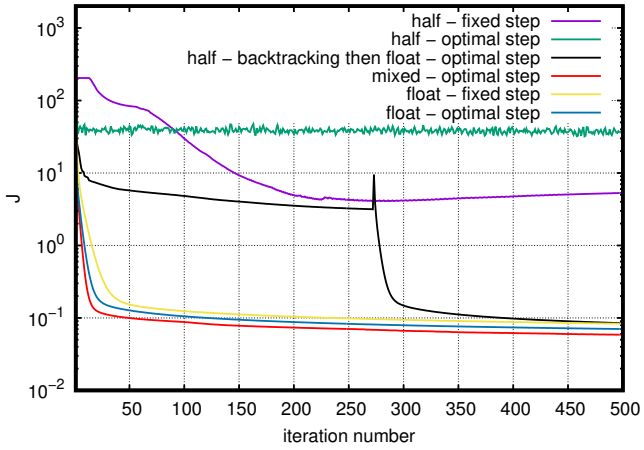
Fig. 6: Reconstructions of f with different precisions (the cubic root is displayed for better contrast)

Deconvolution relying on half floats seems also much more sensitive to noise levels. By comparing figures 7a and 7b, we notice different behaviors in the half-float implementations. The fixed-step version seems noisy with a 16dB SNR but not with a 37dB SNR. There are even differences in the noisy-shaped curves' behavior: they appear to slowly converge on a noisier dataset (16dB SNR). This may be explained by some form of dithering.

In any case, you must put extra care when using half floats as their range is very limited. This issue arises when using cuFFT uses the Fourier domain do compute convolutions. As cuFFT performs non-normalized transforms, half float numbers are easily overflowed. Infinite values will appear in the DFT and lead to wrong results. If you try to first divide your image values by the number of elements, you



(a) Image generated with a 16dB SNR



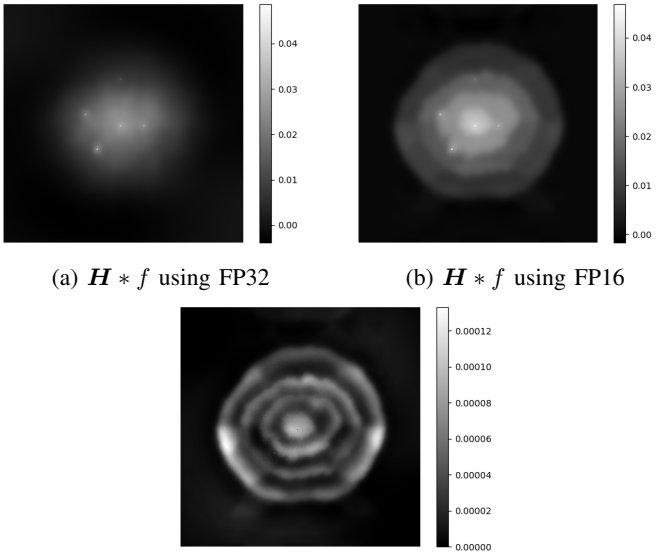
(b) Image generated with a 37dB SNR

Fig. 7: Criterion value across iterations

will underflow and set most values to zero (depending on the size of your data). A solution is to pre-divide by the square root of the number of elements, do the cuFFT, then re-divide by the square root of the number of elements.

Even with this extra care, it was not possible to rely on the convolution in the descent algorithm shown in III. The criterion value is indeed imprecise. This can clearly be seen in figure 7 when stepping from half to single precision in the "half then float" method. Even with the same image, the criterion significantly differs depending on the precision used for its computation. Anyway, using the best image computed with FP16 as an initializer for the FP32 method is slightly better than using a zero-filled image. It is, however, equivalent to an image found after only a few iterations in single precision.

It is unclear why the images produced with the optimal step method using half floats visually give rather good results. Across the iterations, FP16 images do appear to be better even though the criterion does not decreases (even we computed in FP32). The problem might be in the definition of "visually better". Multiple images hold the same criterion value but some may "look" closer to the reconstruction. We



(a) $\boldsymbol{H} * f$ using FP32 (b) $\boldsymbol{H} * f$ using FP16

(c) Squared difference between FP16 and FP32 results

Fig. 8: Convolution errors with our dataset

are currently investigating this issue.

More generally, a dataset involving a smaller kernel may mitigate many problems. When possible, it seems appropriate to use half floats only for storage and convert them on-the-fly as single floats to benefit from lighter data transfer and reasonable accuracy. We can observe on figure 7 that the "mixed" curve has similar performance as the float-only implementation. This kind of strategy is in fact used by Nvidia in their Tensor Cores[7]. In conclusion, the switch from FP32 to FP16 should be done carefully.

VI. Conclusion

In this paper, we have observed the non-negligible loss of precision for 2D convolution using half-precision arithmetic on GPUs. We have pointed out that for limited convolution kernel sizes, a good compromise between acceleration and calculation error is to use a storage in half and a computation in single.

Then, we incorporated half precision arithmetic within a complex application: optimization for image reconstruction in radio astronomy with a kernel convolution of the same size as the 2D images. We tried several methods to choose the step's size in the gradient descent and achieved good visual results. Even though a good convergence does not seem to be achieved using solely half precision, relying on it only for storage but performing computations as floats makes the algorithm converge.

References

[1] IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008*, pages 1–70, August 2008.

[2] L. Lacassagne, D. Etiemble, and S. A. O. Kablia. 16-bit floating point instructions for embedded multimedia applications. In *Seventh International Workshop on Computer Architecture for Machine Perception (CAMP'05)*, pages 198–203, July 2005.

[3] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep Learning with Limited Numerical Precision. *arXiv:1502.02551 [cs, stat]*, February 2015. arXiv: 1502.02551.

[4] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications. *arXiv:1412.7024 [cs]*, December 2014. arXiv: 1412.7024.

[5] N. M. Ho and W. F. Wong. Exploiting half precision arithmetic in Nvidia GPUs. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, September 2017.

[6] Nvidia. GP100 Pascal Whitepaper, https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf.

[7] Nvidia. Volta V100 whitepaper, http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf.

[8] P. Luszczek, J. Kurzak, I. Yamazaki, and J. Dongarra. Towards numerical benchmark for half-precision floating point arithmetic. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–5, September 2017.

[9] Clemens Maaß, Matthias Baer, and Marc Kachelrieß. CT image reconstruction with half precision floating-point values. *Medical Physics*, 38(S1):S95–S105, July 2011.

[10] O. Fialka and M. Cadik. FFT and Convolution Performance in Image Filtering on GPU. In *Tenth International Conference on Information Visualisation (IV'06)*, pages 609–614, July 2006.

[11] Gilles Perrot, Stéphane Domas, and Raphaël Couturier. An optimized GPU-based 2d convolution implementation: AN OPTIMIZED GPU-BASED 2d CONVOLUTION IMPLEMENTATION. *Concurrency and Computation: Practice and Experience*, 28(16):4291–4304, November 2016.

[12] Pavan Yalamanchili, Umar Arshad, Zakiuddin Mohammed, Pradeep Garigipati, Peter Entschev, Brian Kloppenborg, James Malcolm, and John Melonakos. *ArrayFire - A high performance software library for parallel computing with an easy-to-use API*. AccelerEyes, Atlanta, 2015.

[13] Jérôme Idier and Laure Blanc-Féraud. Bayesian Approach to Inverse Problems. pages 141–167. January 2010.

[14] Jonathan Richard Shewchuk. *An introduction to the conjugate gradient method without the agonizing pain*. Carnegie-Mellon University. Department of Computer Science, 1994.

[15] J. J. Dongarra, Jeremy Du Croz, Sven Hammarling, and I. S. Duff. A Set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.*, 16(1):1–17, March 1990.

[16] Deep Learning Frameworks, https://developer.nvidia.com/deep-learning-frameworks. *NVIDIA Developer*, April 2016.

[17] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cuDNN: Efficient Primitives for Deep Learning. *arXiv:1410.0759 [cs]*, October 2014. arXiv: 1410.0759.

[18] Jan E. Noordam and Oleg M. Smirnov. The MeqTrees software system and its use for third-generation calibration of radio interferometers. *Astronomy & Astrophysics*, 524:A61, December 2010. arXiv: 1101.1745.