# NONUNIFORM FAST FOURIER TRANSFORM ON TPUS

*Tianjian Lu*[⋆] *Thibault Marin*[†] *Yue Zhuo*[†] *Yi-Fan Chen*[⋆] *Chao Ma*[†]

[⋆] Google Research
[†]Gordon Center for Medical Imaging, Massachusetts General Hospital, Harvard Medical School

## ABSTRACT

This work presents a parallel algorithm for implementing the nonuniform Fast Fourier transform (NUFFT) on Google's Tensor Processing Units (TPUs). TPU is a hardware accelerator originally designed for deep learning applications. NUFFT is considered as the main computation bottleneck in magnetic resonance (MR) image reconstruction when $k$-space data are sampled on a nonuniform grid. The computation of NUFFT consists of three operations: an apodization, an FFT, and an interpolation, all being formulated as tensor operations in order to fully utilize TPU's strength in matrix multiplications. The implementation is with TensorFlow. Numerical examples show 20x $\sim$ 80x acceleration of NUFFT on a single-card TPU compared to CPU implementations. The strong scaling analysis shows a close-to-linear scaling of NUFFT on up to 64 TPU cores. The proposed implementation of NUFFT on TPUs is promising in accelerating MR image reconstruction and achieving practical runtime for clinical applications.

***Index Terms*—** Nonuniform fast Fourier transform, NUFFT, Magnetic Resonance Imaging, Parallel Computing, TensorFlow, Tensor Processing Unit

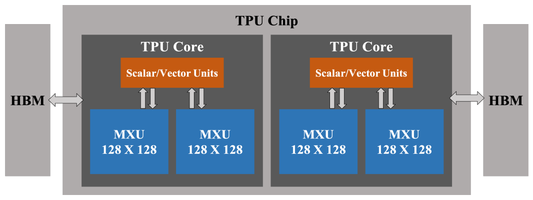
## 1. INTRODUCTION

Nonuniform Fast Fourier transform (NUFFT) [1, 2] is a powerful fast algorithm widely used in a number of scientific and engineering applications such as medical imaging [3], numerical solutions to differential and integral equations [4], and synthetic aperture radar (SAR) imaging [5]. NUFFT enables a variety of applications that require unequally spaced data while inheriting the computation efficiency from FFT [6]. The state-of-the-art image reconstruction methods[7, 8] in magnetic resonance imaging (MRI) that build upon large-scale, iterative, optimization algorithms have an extensive usage of NUFFT when k-space data are sampled on a nonuniform grid. However, the lag time of these advanced MR image reconstruction algorithms, defined as the delay between data acquisition and image display, can often be unacceptable for clinical use, largely due to the bottleneck in computing NUFFT.

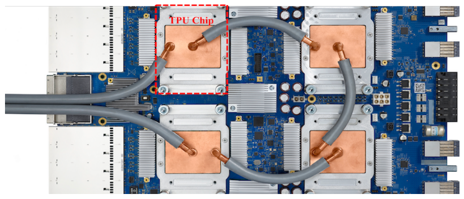
Graphics Processing Units (GPUs) have been extensively studied to accelerate NUFFT [9, 10]. In this work, we propose an alternative implementation of NUFFT on Google's Tensor Processing Units (TPUs) [11, 12]. TPU is an application-specific integrated circuit (ASIC) to run cutting-edge ML models on Google Cloud [11]. Figure 1 shows the TPU chip and unit (or board): one TPU unit contains four chips; each chip has two cores; and each core contains the scalar, vector, and matrix units (MXU). The chips are connected directly through dedicated, high-speed, and low-latency interconnects, bypassing the host CPU and networking resources. MXU provides the bulk of the compute power, which handles 16 K multiply-accumulate (MAC) operations in one single clock cycle. Each TPU core has 16 GiB high-bandwidth memory (HBM). Recently, TPU has been studied to tackle large-scale scientific computing problems [13, 14]. In particular, TPUs have been successfully used to accelerate magnetic resonance (MR) image reconstruction with non-Cartesian sampling [15]. The image reconstruction in [15] has the nonuniform Fourier transform formulated as dense matrix multiplications or discrete Fourier transform (DFT). For a $N$-point transform, the computation complexity of DFT is $\mathcal{O}(N^2)$, whereas it is $\mathcal{O}(N \log N)$ for FFT. The proposed implementation of NUFFT on TPUs brings in the reduction of computation complexity and can further accelerate MR image reconstruction on TPUs.

The computation of NUFFT consists of three operations: an apodization, an FFT, and an interpolation, all being formulated as tensor operations in order to fully utilize TPU's strength in matrix multiplications. The apodization operation is point-wise over the image intensities. With a data decomposition applied to the spatial coordinates, it can be performed within individual cores and completely in parallel. The parallel algorithm of FFT on TPUs consists of two major operations, the local transform based on the famous Cooley-Tukey algorithm [6] on individual cores and the phase adjustment, the details of which can be found in [14]. The interpolation operation builds upon a convolution, which is formulated as tensor contractions between kernel-function values and patches extracted from the Fourier transform of an oversampled image. In this work, the Kaiser-Bessel function [16]

Correspondence to: Y.-F. Chen (yifanchen@google.com) and C. Ma (cma5@mgh.harvard.edu)



(a)



(b)

**Fig. 1**: The TPU v3 (a) chip, (b) unit (or board). One TPU board has four chips; each chip contains two cores; and a TPU v3 Pod in a data center contains 2048 cores.

is selected as the convolution kernel. The implementation of NUFFT on TPUs is with TensorFlow. The strong scaling analysis demonstrates close-to-linear parallel efficiency of the proposed algorithm.

## 2. METHODS

The discrete Fourier transform with unequally sampled data can be expressed as

$$s\left(k_{x,m}, k_{y,m}\right) = \sum_{n=1}^{N} \rho_n e^{-i2\pi(k_{x,m}x_n + k_{y,m}y_n)} \tag{1}$$

where $(k_{x,m}, k_{y,m}), m = 1, 2, \cdots, M$ represents the $k$-space coordinates on a nonuniform grid, $(x_n, y_n), n = 1, 2, \cdots, N$ represents the spatial coordinates on a uniform grid, and $\rho_n$ denotes the image intensity on grid $(x_n, y_n)$.

Direct computation of Eq. (1) for all k-space samples has computation complexity of $\mathcal{O}(MN)$. NUFFT seeks an approximate solution to Eq. (1) while leveraging the high computation efficiency of FFT, which can be written concisely in the following matrix/vector mulplification form:

$$\mathbf{s} = \mathbf{CFD}\boldsymbol{\rho}, \tag{2}$$

where $\mathbf{D}$ is the apodization operator, $\mathbf{F}$ denotes the FFT operator, and $\mathbf{C}$ represents the interpolation operator.

### 2.1. Apodization

The apodization operation is defined as

$$\{\mathbf{D}\boldsymbol{\rho}\}_n = \frac{\rho_n}{\mathrm{d}(x_n)\mathrm{d}(y_n)} \tag{3}$$

where $\mathrm{d}(\cdot)$ represents the inverse Fourier transform of a convolution kernel used in the interpolation operation. In this work, we choose the Kaiser-Bessel function as the kernel [16]. The apodization operation is point-wise over the image intensities $\rho_n$. With a data decomposition applied to the spatial coordinates $(x_n, y_n)$, the apodization operation can be parallely performed within individual TPU cores.

### 2.2. FFT

FFT in NUFFT operates on a zero-padded image $\rho_{\text{pad}}$ of size $\alpha N_x \times \alpha N_y$, which can be obtained by

$$\rho_{\text{pad}} = \begin{cases} \rho_n & \text{if } |x_n| \le \frac{N_x}{2} \text{ and } |y_n| \le \frac{N_y}{2} \\ 0 & \text{elsewhere} \end{cases}. \tag{4}$$

The parallel algorithm of FFT on TPUs consists of two major operations, a local transform based on the Cooley-Tukey algorithm and the phase adjustment. Higher dimension FFT builds upon the one-dimension (1D) transform, the formulation of which starts with

$$\Gamma_k \triangleq \sum_{q=0}^{N_q-1} \gamma_q e^{-i2\pi\dfrac{qk}{N_q}} \tag{5}$$

and $\gamma_q$ represents the equally-spaced image intensities. The global index $q$ in Equation (5) can be expressed as

$$q = Pl + \nu, \tag{6}$$

where $l = 0, 1, \cdots, \frac{N_q}{P} - 1$, $\nu = 0, 1, \cdots, P-1$, and $P$ denotes the number of TPU cores used to perform the 1D transform. After substituting the global index, Eq. (5) can be rewritten as

$$\Gamma_k \triangleq \sum_{q=0}^{N_q-1} \gamma_{(Pl+\nu)} e^{-i2\pi\dfrac{(Pl+\nu)k}{N_q}} \tag{7}$$

$$= \sum_{\nu=0}^{P-1} e^{-i2\pi\dfrac{\nu k}{N_q}} \left( \sum_{l=0}^{\frac{N_q}{P}-1} \gamma_{(Pl+\nu)} e^{-i2\pi\dfrac{lk}{\frac{N_q}{P}}} \right). \tag{8}$$

It is shown in Eq. (8) that

$$\widetilde{\Gamma}_k = \sum_{l=0}^{\frac{N}{P}-1} \gamma_{(Pl+\beta)} e^{-i2\pi\dfrac{lk}{\frac{N}{P}}} \tag{9}$$

can be computed with the Cooley-Tukey algorithm locally on individual cores and completely in parallel. The final results can be obtained by summing the local transform over all the TPU cores with the phase adjustment term $e^{-i2\pi\frac{\nu k}{N_q}}$, with $\nu$ representing the TPU core index. See [14] for more details of parallel implementation of FFT on TPUs.

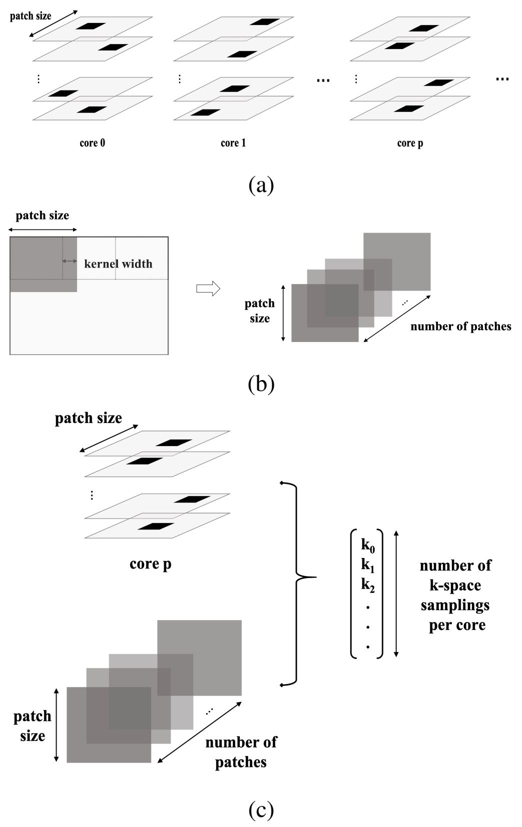


**Fig. 2**: The interpolation operation on TPUs is formulated as tensor contractions: (a) the kernel-function values are pre-computed on the CPU host; the data decomposition is applied in the $k$-space such that each TPU core contains a portion of the kernel-function values; zero-padding is required and the nonzero values are highlighted in dark; (b) patches are extracted from the Fourier transform results of a padded image with a depth-wise convolution; (c) tensor contraction is applied between the kernel-function values and the extracted patches, followed by a patch-selection with Boolean mask and a dimension reduction.

### 2.3. Interpolation

The interpolation operation can be written as

$$\{\mathbf{C}\boldsymbol{\xi}\}_m = \sum_q \sum_p C\left(\frac{p - \alpha k_{x,m}}{w/2}\right) C\left(\frac{q - \alpha k_{y,m}}{w/2}\right) \boldsymbol{\xi}_{p,q}$$

$$p : |p - k_{x,m}| \le \frac{w}{2} \text{ and } q : |q - k_{y,m}| \le \frac{w}{2}$$

$$k_{x,m} \in \left[-\frac{N_x}{2}, \frac{N_x}{2}\right) \text{ and } k_{y,m} \in \left[-\frac{N_y}{2}, \frac{N_y}{2}\right), \qquad (10)$$

where $\xi$ represents the Fourier transform of a zero-padded image and $C(\cdot)$ denotes the kernel function. The interpolation operation is formulated as tensor contractions as shown in Fig. 2. The kernel-function values are computed on the CPU host. As shown in Fig. 2(a), the kernel-function values corresponding to one $k$-space sampling point are stored as a 2D slice of a 3D tensor. The size of the 2D slice is larger than the kernel width because zero-padding is applied to the computed kernel-function values. Prior to the zero-padding, the 2D tensor containing the nonzero kernel-function values has the shape $w \times w$ with $w$ denoting the kernel width. The zero-padding enables the formulation of the interpolation operation as tensor contractions, which are very efficient on TPUs. The Fourier transform of the zero-padded image, or $\xi$ in Equation (10) is extracted into patches. As shown in Fig. 2(b), the size of an extracted patch is the same as that of a 2D slice containing the kernel-function values. The patch extraction is implemented with the depth-wise convolution `tf.nn.depthwise_conv2d`. As shown in Fig. 2(c), the interpolation operation is computed as a tensor contraction with `tf.einsum` between the kernel-function values and the extracted patches. With the data decomposition applied to the $k$-space, each TPU core contains a portion of the kernel-function values. Each TPU core has the full Fourier transform results of the zero-padded image. The interpolation operation can be performed locally on individual TPU cores and completely in parallel.

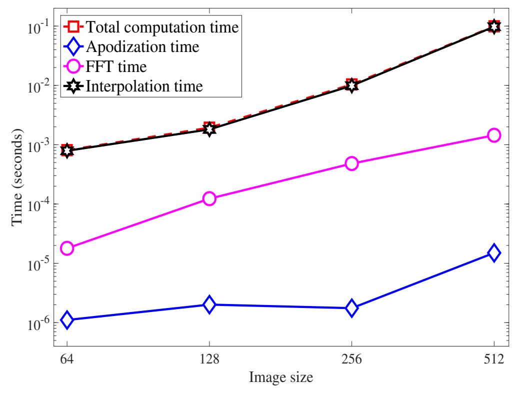


**Fig. 3**: The breakdown of the total computation time of NUFFT on eight TPU cores (or one TPU unit). The image size varies from $64 \times 64$ to $512 \times 512$.

## 3. RESULTS

Figure 3 shows the breakdown of the total computation time in term of the apodization, the FFT, and the interpolation. In this example, the image size increases from $64 \times 64$ to $512 \times 512$ but the number of cores remains as eight (one TPU unit). The width of the Kaiser-Bessel kernel is chosen as four and the oversampling factor is two. It can be seen from Fig. 3 that the interpolation operation is the most expensive operation of NUFFT in terms of the computation time. The FFT operation takes a very small portion of the total computation time: for the image size of $256 \times 256$, the time on FFT is $4.64\%$ of the total computation time; and it is $1.45\%$ for the image size of $512 \times 512$. Given the portion in terms of both the computation

**Table 1**: Computation time of NUFFT on two types of hardware: CPU–Intel(R) Xeon(R) Silver 4110 8-core 2.10 GHz and TPU–one TPU v3 unit (eight cores).

| Time (ms) | | FFT | | Interpolation | | Total | |
|---|---|---|---|---|---|---|---|
| Hardware | | CPU | TPU | CPU | TPU | CPU | TPU |
| Image Size | 64 x 64 | 0.56 | 0.02 | 63.37 | 0.79 | 64.06 | 0.81 |
| | 128 x 128 | 2.06 | 0.12 | 59.30 | 1.81 | 61.53 | 1.94 |
| | 256 x 256 | 11.80 | 0.48 | 240.33 | 9.90 | 252.62 | 10.38 |

time and the memory usage that FFT takes in NUFFT and the large capacity of HBM on each TPU core, we have each core perform the FFT operation over a non-partitioned image.

As shown in Table 1, we perform a preliminary comparison for NUFFT on CPU and TPU. NUFFT on CPU was implemented with SigPy [17]. The CPU used for the comparison is Intel(R) Xeon(R) Silver 4110 8-core 2.10 GHz. The computation time of FFT on one TPU unit (eight cores) is much smaller than that on CPU for all three examples in Table 1. The interpolation is the computation bottleneck of NUFFT for both types of hardware. Percentage-wise and for the case of $256 \times 256$, NUFFT on TPU takes $95.4\%$ of the total computation time and that are $95.1\%$ for CPU. Our current implementation of NUFFT on TPUs achieved 20x $\sim$ 80x acceleration compared to CPU implementations.

Figure 4 shows the strong scaling analysis for NUFFT on TPUs: the image size remains as $1024 \times 1024$ and the number of TPU cores increases from 16 to 128. The number of sampling points in $k$-space is 1,647,616. The width of the Kaiser-Bessel kernel is chosen as four and the oversampling factor is two. The speed-up in Fig. 4 is defined as:

$$\text{speed-up} = \frac{T_{16}}{T_{N_{\text{core}}}}, \tag{11}$$

where $T_{16}$ denotes the computation time with 16 TPU cores and $T_{N_{\text{core}}}$ represents the computation time with $N_{\text{core}}$ cores. It can be seen from Fig. 4 that a close-to-linear scaling is achieved up to 64 cores. The HBM usage for the case with 64 TPU cores is 1.57 GB. The problem size is considered as small for more than 64 cores, which also explains why the gain of speed-up starts saturating. The total computation time is 796.01 ms by using 64 TPU cores and 160.27 ms with 256 cores.

## 4. DISCUSSION & CONCLUSION

In this proof-of-concept study, we proposed and implemented a parallel algorithm of NUFFT on TPUs, the domain-specific hardware originally developed for deep learning applications. In order to fully utilize TPU's strength in matrix multiplications, the computation of NUFFT is formulated as tensor

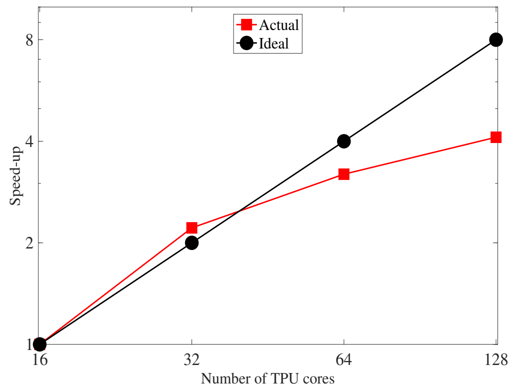


**Fig. 4**: The speed-up of NUFFT on TPUs for an image of size $1024 \times 1024$ and with up to 128 TPU cores. The number of sampling points in $k$-space is 1,647,616.

operations. Numerical examples show that the proposed implementation 20x $\sim$ 80x acceleration compared to CPU implementations. More importantly, the strong scaling analysis shows a close-to-linear scaling of NUFFT on up to 64 TPU cores, indicating that MR image reconstructions can be efficiently accelerated with multiple TPUs. Several future work are on going for accelerated MR image reconstruction on TPUs, including more efficient implementation of the interpolation operator, the implementation of the adjoint operator of NUFFT, and the implementation of NUFFT-based interative image reconstruction algorithms, e.g., ADMM. Once accomplished, we also plan to perform systematic comparsions of image reconstruction time on CPUs, GPUs and TPUs.

In conclusion, we propose an implementation of NUFFT on Google's TPUs. The proposed method is promising in accelerating MR image reconstruction and achieving practical runtime for clinical applications.

## 5. COMPLIANCE WITH ETHICAL STANDARDS

This is a numerical simulation study for which no ethical approval was required.

## 6. ACKNOWLEDGMENT

Thibault Marin, Yue Zhuo, and Chao Ma were supported in part by the National Institute of Health under award: T32EB013180, R01CA165221, R01HL118261, R01HL137230 and P41EB022544.

Tianjian Lu and Yi-Fan Chen are Google employees. Tianjian Lu and Yi-Fan Chen would like to thank Tao Wang and Blake Hechtman at Google for valuable discussions and helpful comments.

## 7. REFERENCES

[1] Qing Huo Liu and Nhu Nguyen, "An accurate algorithm for nonuniform fast fourier transforms (NUFFT's)," *IEEE Microwave and guided wave letters*, vol. 8, no. 1, pp. 18–20, 1998.

[2] Jeffrey A. Fessler and Bradley P. Sutton, "Nonuniform fast Fourier transforms using min-max interpolation," *IEEE Transactions on Signal Processing*, vol. 51, no. 2, pp. 560–574, 2003.

[3] Jeffrey A Fessler, "On NUFFT-based gridding for non-cartesian MRI," *Journal of magnetic resonance*, vol. 188, no. 2, pp. 191–195, 2007.

[4] Qing Huo Liu, Xue Min Xu, Bo Tian, and Zhong Qing Zhang, "Applications of nonuniform fast transform algorithms in numerical solutions of differential and integral equations," *IEEE Transactions on geoscience and remote sensing*, vol. 38, no. 4, pp. 1551–1560, 2000.

[5] B Subiza, E Gimeno-Nieves, JM Lopez-Sanchez, and J Fortuny-Guasch, "An approach to SAR imaging by means of non-uniform FFTs," in *IGARSS 2003. 2003 IEEE International Geoscience and Remote Sensing Symposium. Proceedings (IEEE Cat. No. 03CH37477)*. IEEE, 2003, vol. 6, pp. 4089–4091.

[6] James W Cooley and John W Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[7] Michael Lustig, David L Donoho, Juan M Santos, and John M Pauly, "Compressed sensing MRI," *IEEE signal processing magazine*, vol. 25, no. 2, pp. 72–82, 2008.

[8] Zhi-Pei Liang, "Spatiotemporal imaging with partially separable functions," in *2007 4th IEEE International Symposium on Biomedical Imaging: From Nano to Macro*. IEEE, 2007, pp. 988–991.

[9] Sam S Stone, Justin P Haldar, Stephanie C Tsao, BP Sutton, Z-P Liang, et al., "Accelerating advanced MRI reconstructions on GPUs," *Journal of parallel and distributed computing*, vol. 68, no. 10, pp. 1307–1318, 2008.

[10] Yue Zhuo, Xiao-Long Wu, Justin P Haldar, Wen-mei Hwu, Zhi-Pei Liang, and Bradley P Sutton, "Accelerating iterative field-compensated MR image reconstruction on GPUs," in *2010 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*. IEEE, 2010, pp. 820–823.

[11] "Cloud TPUs @ONLINE," `https://cloud.google.com/tpu/`.

[12] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al., "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 1–12.

[13] Kun Yang, Yi-Fan Chen, Georgios Roumpos, Chris Colby, and John Anderson, "High performance Monte Carlo simulation of Ising model on TPU clusters," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2019, SC '19, pp. 83:1–83:15, ACM.

[14] Tianjian Lu, Yi-Fan Chen, Blake Hechtman, Tao Wang, and John Anderson, "Large-scale discrete Fourier transform on TPUs," *arXiv preprint arXiv:2002.03260*, 2020.

[15] Tianjian Lu, Thibault Marin, Yue Zhuo, Yi-Fan Chen, and Chao Ma, "Accelerating MRI reconstruction on TPUs," *arXiv preprint arXiv:2006.14080*, 2020.

[16] Robert M Lewitt, "Multidimensional digital image representations using generalized Kaiser–Bessel window functions," *JOSA A*, vol. 7, no. 10, pp. 1834–1846, 1990.

[17] F Ong and M Lustig, "Sigpy: a python package for high performance iterative reconstruction," in *Proceedings of the ISMRM 27th Annual Meeting, Montreal, Quebec, Canada*, 2019, vol. 4819.