Accelerating spherical harmonic transforms for a large number of sky maps

CHI TIAN,1 SIYU LI,2 AND HAO LIU∗1,2

1School of Physics and Optoelectronics Engineering, Anhui University, 111 Jiulong Road, Hefei, Anhui, China 230601.
2Key Laboratory of Particle and Astrophysics, Institute of High Energy Physics, CAS, 19B YuQuan Road, Beijing, China, 100049.

ABSTRACT

The spherical harmonic transform is a powerful tool in the analysis of spherical data sets, such as the cosmic microwave background data. In this work, we present a new scheme for the spherical harmonic transforms that supports both CPU and GPU computations, which is especially efficient on a large number of sky maps. By comparing our implementation with the standard Libsharp-HEALPix program, we demonstrate 2-10 times speedup for the CPU implementation, and up to 30 times speedup when a state-of-the-art GPU is employed. This new scheme’s software package is available via an open source GitHub repository.

1. INTRODUCTION

The spherical harmonic transform (SHT) decomposes signals on a sphere into sets of spherical harmonic coefficients based on the spherical harmonic functions. As an analog to the Fourier transform, the SHT extracts scale-dependent features on the sphere and has been extensively used in analyzing spherical data in various areas of science, including cosmology and the cosmic microwave background (CMB) science. For example, by performing SHTs on the CMB anisotropy data and fitting with the concordance cosmological model (ΛCDM), we are able to make precise estimations about the cosmological parameters. The pipeline for processing CMB polarization data with SHT is also crucial for detecting the primordial gravitational waves. Recent developments in cosmological surveys in CMB temperature and polarization anisotropies (Keating et al. 2003; The Planck Collaboration 2006; Rubiño-Martín et al. 2010; Niemack et al. 2010; Keating et al. 2011; Austermann et al. 2012; Hazumi et al. 2012; Abazajian et al. 2016; Li et al. 2018; Ade et al. 2019) pose new challenges to data analysis, which often requires processing a large amount of CMB mock data. Therefore, it is important to reduce the time cost of SHTs on numerous spherical maps.

The optimizations for the SHT have been intensively studied in the past years. Optimized codes, such as Libsharp (Reinecke & Seljebotn 2013) and SHTns (Schaeffer 2013) have been used in various cosmological studies. However, it is difficult to further accelerate the SHT on a single sky map, because the most time-consuming part of the SHT is effectively a BLAS (basic linear algebra subprograms) level 2 problem, which is not very suitable for high-performance computing (HPC). In this work, we discover that when there are many input maps, the core part of SHT can be updated from BLAS level 2 to level 3, which is not only appropriate for HPC but also has many established solutions. Following this idea, we propose a highly efficient SHT algorithm that is able to process a large number of spherical maps in a batch. We will demonstrate the power of this new scheme by carrying out SHTs on mock CMB temperature and polarization maps with our open source software package — fastSHT1, whose performance will be compared with the SHT toolkit Libsharp, which is the SHT engine of HEALPix (Górski et al. 2005) and Healpy, where the latter provides a python interface.

Also note that the recent developments in the Graphic Processing Units (GPUs) pave the way to further accelerating the traditional spherical data analysis pipeline, which was mostly based on computations with CPUs. Pioneering studies such as Hupca et al. (2012) have demonstrated the power of GPUs in computing the SHT on a single map. In this work, we will explore both the GPU and CPU implementations of our new SHT algorithm with the state-of-the-art computer hardware.

This paper is organized as follows, we first introduce the mathematical basis of the SHT in section 2, and then use CMB temperature and polarization maps to explain in detail how to accelerate SHTs for a large number of spherical maps. The software implementation of the new SHT scheme is presented in section 3, where comprehensive memory usage estimations and performance benchmarks with CPUs and GPUs are included. Then a new iteration scheme based

∗Corresponding author, uste_liuhao@163.com

1 https://github.com/liuhao-cn/fastSHT, and the version corresponding to this work is Chi Tian et al. (2022).
on our software implementation is introduced in section 4, which significantly improve the quality of iterative estimation. Finally, we conclude and discuss in section 5.

2. MATHEMATICAL BASIS

The basic algorithm of fast SHT has been described in a number of literatures (Mollenkamp 1999; Suda & Takami 2002; Doroshkevich et al. 2005; Huffenberger & Wandelt 2010; Reinecke 2011; Fabbian et al. 2012; McEwen et al. 2013). In particular, some tips of acceleration are summarized in Schaeffer (2013). Till the present day, there seems to be no significant changes to these basic aspects of the algorithm, particularly for the single-map computation; thus, for convenience of reading, the basis of the algorithm is briefly summarized below, followed by our new batch SHT scheme.

2.1. SHT of CMB temperature maps

The forward SHT decomposes a scalar field on a sphere, such as the CMB temperature map $T(\theta, \phi)$, into spherical harmonic coefficients $a_{\ell m s}$, following the equation

$$a_{\ell m s} = \sum_{\theta, \phi} T(\theta, \phi) Y_{\ell m}(\theta, \phi) d\sigma,$$

where $\theta$ and $\phi$ are discretized polar and azimuthal angle respectively, and $d\sigma = 4\pi/N_{\text{pix}}$ is the area of one pixel, which is a constant in the HEALPix pixelization scheme.

Considering a series of maps denoted by index $k$, and decomposing $Y_{\ell m}(\theta, \phi)$ into the associate Legendre polynomials $P_{\ell m}[\cos(\theta)]$ with a phase term\(^2\), the above equation becomes

$$a_{\ell m}^k = \sum_{\theta, \phi} T^k(\theta, \phi) P_{\ell m}[\cos(\theta)] e^{-i m \phi} d\sigma$$

$$= \sum_{\theta, \phi} T^k(\theta, \phi) P_{\ell m}[\cos(\theta)] \left[ e^{-i \frac{2\pi m}{N}} e^{-i m \phi_0(\theta)} \right] d\sigma$$

$$= \sum_{\theta} P_{\ell m}[\cos(\theta)] \left[ \sum_{n} T^k(\theta, n) e^{-i \frac{2\pi m n}{N}} \right] e^{-i m \phi_0(\theta)} d\sigma$$

$$= \sum_{\theta} P_{\ell m}[\cos(\theta)] T_m^k(\theta),$$

where $N$ is the number of pixels in the iso-latitude ring located at $\theta$, $n$ is the local index of the pixels inside the ring, and $\phi_0(\theta)$ is the $\phi$-coordinate of the first pixel in each ring. The value $T_m^k(\theta) \equiv \left[ \sum_{n} T^k(\theta, n) e^{-i \frac{2\pi m n}{N}} \right] e^{-i m \phi_0(\theta)}$ is the combination of Fourier transforms of $T(\theta, \phi)$ with constant phase shifts, which can be computed by a forward fast Fourier transform (FFT). The result of FFT is phase-shifted as above and properly rearranged into matrix forms, which is called the “FFT mapping” in the following part of this work. The FFT mapping usually takes about 10-15% of the total computation time, whereas the most time-consuming part is the last row of the above equation. If there is only one input map, the operation in the last row effectively involve only matrix-to-vector multiplications, since the subscript $m$ does not participate in the computation. This kind of task is called a BLAS level 2 problem. Unfortunately, it is well known that a BLAS level 2 problem cannot be significantly accelerated, whereas a BLAS level 3 problem, such as a matrix-to-matrix multiplication (MM), can be effectively optimized by fully exploiting the cache-based architecture (see Goto & Van De Geijn (2008)).

When there are a large number of spherical maps, the last row of eq. (2) becomes a matrix-to-matrix multiplication that contracts the $\theta$-dimension, which looks like

$$\left( a_{\ell}^k \right)_m = \left( P_{\ell}^m \cdot T_{\theta}^k \right)_m.$$  \hspace{1cm} (3)

$m$ is written outside the brackets to indicate that it is not part of the matrix multiplication. This reformulation suggests that matrix multiplications could be employed to accelerate SHTs of a batch of sky maps.

Similarly, the backward transform is given by equations below:

$$T^k(\theta, \phi) = \sum_{\ell m} a_{\ell m}^k P_{\ell m}[\cos(\theta)] e^{i m \phi}$$

$$= \sum_{\ell m} a_{\ell m}^k P_{\ell m}[\cos(\theta)] e^{i \frac{2\pi m n}{N}} e^{i m \phi_0(\theta)}$$

$$= \sum_{m=0}^{l_{\text{max}}} \sum_{l=0}^{l_{\text{max}}} e^{i \frac{2\pi m n}{N}} e^{i m \phi_0(\theta)} a_{\ell m}^k P_{\ell m}[\cos(\theta)]$$

$$= \sum_{m=0}^{l_{\text{max}}} T_m^k(\theta) e^{i \frac{2\pi m n}{N}},$$

where $T_m^k(\theta) = \left[ e^{i m \phi_0(\theta)} \sum_{l=0}^{l_{\text{max}}} a_{\ell m}^k P_{\ell m}[\cos(\theta)] \right]$ should be computed by another matrix multiplication that contracts $\ell$, which looks like

$$\left( T_{\theta}^k \right)_m = \left( a_{\ell}^k \cdot P_{\ell}^m \right)_m.$$  \hspace{1cm} (5)

and then the last row of eq. (4) is computed by a backward-FFT ring by ring, with $T_m^k(\theta)$ being the input FFT-coefficients.

To summarize, for a batch of input sky maps, the forward SHT can be accomplished by computing the FFT, FFT-mapping, and MMs; and the backward SHT can be computed in reverse order by performing the MM, FFT-mapping, and FFT. Compared to the regular scheme that processes the maps one by one, this newly designed pipeline fully exploits

\(^2\) Here the associated Legendre polynomials are usually re-normalized to the spherical harmonics as $|P_{\ell m}[\cos(\theta)]| = |Y_{\ell m}(\theta, \phi)|$, and this kind of re-normalized $P_{\ell m}$ are also referred to as $\lambda_{\ell m}$ and shortened as “lam” in several source codes, e.g., HEALPix.
the optimizations from matrix multiplications and can therefore be more efficient. Moreover, this new scheme also allows more effective usage of the GPU and enables a much bigger room for acceleration.

2.2. Simplified matrix form

Eq. (3) and eq. (5) are the cores of our new SHT method, and they can be further simplified by defining the following matrices:

\[ A_m = (a^E_k)_m, \quad T_m = (T^E_k)_m, \quad P_m = (P^E_k)_m. \]  

(6)

Note that these matrices are usually non-square. The spherical harmonic transforms are then given by the following matrix multiplications at each \( m \), with the convention that the superscript of the first matrix is contracted with the subscript of the second matrix.

\[ A_m = P_m T_m \]

(7)

\[ T_m = P^t_m A_m. \]

Then \( T_m \) is connected to the temperature map by the FFT.

2.3. SHT of the CMB polarization maps

In the context of rotations on the sphere, the \( Q \) and \( U \)-Stokes parameters can be decomposed into spin ±2 spherical harmonics (Zaldarriaga & Seljak 1997; Zaldarriaga 1998) as

\[ Q(\hat{n}) \pm iU(\hat{n}) = \sum_{l,m} a_{\pm2,l,m} Y_{lm}(\hat{n}), \]

(8)

where \( \pm2 Y_{lm}(\hat{n}) \) are the spin ±2 spherical harmonics. Similar to eq. (6–7), the spin ±2 spherical harmonic coefficients \( a_{\pm2,l,m} \) are given by the following matrix multiplications:

\[ A_{\pm2,m} = F_{\pm2,m} (Q_m \pm iU_m), \]

(9)

where \( Q_m \) and \( U_m \) are the phase shifted Fourier transforms of the \( Q \) and \( U \)-Stokes parameters of the \( k \)-th map at the \( i \)-th ring, similar to \( T_m \); and \( F_{\pm2,m} \) are the spin±2 spherical harmonics without the phase term, so \( F_{\pm2,m} \) are real valued matrices.

Because the \( E \) - and \( B \) -mode spherical harmonic coefficients are defined as

\[ a_{E,l,m} = -(a_{2,l,m} + a_{-2,l,m})/2, \]

\[ a_{B,l,m} = i(a_{2,l,m} - a_{-2,l,m})/2, \]

(10)

we get

\[ A^E_m = \frac{1}{2} [F_{+2,m} Q_m + F_{-2,m} (iU_m) + F_{-2,m} Q_m - F_{+2,m} (iU_m)] \]

(11)

\[ A^B_m = \frac{1}{2} [F_{+2,m} Q_m + F_{-2,m} (iU_m) - F_{-2,m} Q_m + F_{+2,m} (iU_m)]. \]

By defining

\[ F_{+,m} = -(F_{+2,m} + F_{-2,m})/2 \]

\[ F_{-,m} = -(F_{+2,m} - F_{-2,m})/2, \]

eq. (11) can be simplified to:

\[ A^E_m = F_{+m} Q_m + F_{-m} (iU_m) \]

\[ iA^B_m = F_{-m} Q_m + F_{+m} (iU_m), \]

which can be further simplified to the following matrix form:

\[ \begin{pmatrix} A^E_m \\ iA^B_m \end{pmatrix} = \begin{pmatrix} F_+ & F_- \\ F_- & F_+ \end{pmatrix} \begin{pmatrix} Q_m \\ iU_m \end{pmatrix}. \]

(14)

From eq. (14), we immediately get the backward transform as:

\[ \begin{pmatrix} Q_m \\ iU_m \end{pmatrix} = \begin{pmatrix} F_+ & F_- \\ F_- & F_+ \end{pmatrix}^t \begin{pmatrix} A^E_m \\ iA^B_m \end{pmatrix}. \]

(15)

Therefore, \( iA^B_m \) and \( iU_m \) are more natural than \( A^E_m \) and \( U_m \) in the spherical harmonic transforms of polarized data.

From eqs. (14–15), it is also convenient to derive pure pixel-domain decompositions of the \( E \)- and \( B \)-modes:

\[ \begin{pmatrix} Q_m \\ iU_m \end{pmatrix}^{E,B} = \begin{pmatrix} F_+ & F_- \\ F_- & F_+ \end{pmatrix}^t \begin{pmatrix} A^E_m \\ iA^B_m \end{pmatrix} \]

(16)

where superscripts \( E,B \) means to keep only the \( E \)- and \( B \)-mode components, and \( \lambda^{E,B} \) is a diagonal matrix with either 0 or 1 along the diagonal line. For \( \lambda^E \), only the first half of the diagonal line is equal to 1, and for \( \lambda^B \), only the second half of the diagonal line is equal to 1.

The above form might be the most elegant representation of the essence of \( E \)- and \( B \)-modes, because it shows that, the only difference between the \( E \)- and \( B \)-modes is to divide the identity matrix \( I \) into two parts and keep either the first or the second part non-zero.

Apparently, eqs. (14–16) are block form representations of larger matrices, and the multiplication of each pair of elements means another matrix multiplication. The combined larger matrices can be written as

\[ S_m = \begin{pmatrix} Q_m \\ iU_m \end{pmatrix}, \quad F_m = \begin{pmatrix} F_+ & F_- \\ F_- & F_+ \end{pmatrix}, \quad A_m = \begin{pmatrix} A^E_m \\ iA^B_m \end{pmatrix}, \]

(17)

then the SHTs of the polarized data can be simply computed by the following matrix multiplications:

\[ A_m = F_m S_m, \]

\[ S_m = F_m^t A_m, \]

(18)
where $\mathcal{F}_m$ is a real matrix but might be non-square. When an EB-decomposition is required, it can be done as

$$S_{m}^{E,B} = (\mathcal{F}_m^E \Lambda^{E,B} \mathcal{F}_m) S_m,$$  \hspace{1cm} (19)

where $\mathcal{F}_m$, $\mathcal{F}_m^E$ and $\Lambda^{E,B}$ are both real, diagonal and contain only 0 or 1. These equations are enough to handle the spherical harmonic transforms of polarized data in either pixel or harmonic domains.

3. CODE IMPLEMENTATION AND PERFORMANCE

In this section, we will introduce and benchmark our code, fastSHT, which is an optimized implementation of the new SHT scheme introduced above. Its core part is written in FORTRAN for both the CPU and GPU implementations, and the FORTRAN subroutines are wrapped by the software f90wrap (Kermode 2020) to expose Python interfaces to users. The GPU feature can be optionally turned on/off by the user when compiling the code. For convenience, we will from now on refer to the name fastSHT-CPU as the code compiled only with CPU; and the name fastSHT-GPU as the code compiled with the GPU feature enabled. The fastSHT-CPU code employs the Intel MKL library for the FFT and general matrix multiplication (gemm) to carry out the operations mentioned above; whereas fastSHT-GPU benefits not only from the faster GPU matrix multiplications, but also from the more efficient FFT mappings and iterative operations. The internal loops of fastSHT-GPU are parallelized using OpenACC (Open Accelerators), significantly reducing the coding complexity.

3.1. Implementation detail

**Algorithm 1**: fastSHT algorithm – non-iterative

**Data**: Input sky maps as matrix: $S$

**Result**: Spherical harmonic coefficients: $a_{\ell m}$

```plaintext
1 $F_{\ell m}^k =$ Batch FFT($S$)
2 for $m = 0$ to $\ell_{\text{max}}$ do
3     for $\theta = 0$ to $N_{\text{rings}}$ do
4         for $\ell = 0$ to $\ell_{\text{max}}$ do
5             compute Legendre polynomials $P_m(\ell, \theta)$
6         end
7     for $k = 0$ to $N_{\text{maps}}$ do
8         remap $F_{\ell m}^k$ to $T_{\ell m}(\theta, k)$
9     end
10    $a_{m}(k, \ell) =$ gemm($P_m(\ell, \theta)$, $T_m(\theta, k)$)
11 end
12 end
```

Our fastSHT code is based on the HEALPix pixelization scheme, where the sphere is divided into iso-latitude rings, and then into equal-area pixels. The iso-latitude rings in this pixelization scheme naturally discretize the polar angle $\theta$. We give the pseudo code of our new SHT algorithm in Algorithm 1. The definitions of the auxiliary fields ($P_m$ and $\theta_m$) are in section 2.1, and the variable $N_{\text{rings}}$ is the total number of iso-latitude rings. In addition, to accelerate SHT as much as possible for a large number of sky maps, the following points should be taken into account in the implementation:

1. All SHT operations are done with real numbers to maximize efficiency.
2. The FFT needs to be performed in batch, and any unnecessary memory operations, i.e, read/write operations during the FFT, should be done by the FFT program itself. This means that the FFT implementation should support I/O distances and I/O strides.
3. In a FFT implementation like FFTW or Intel MKL, a plan/handle will be created before computing, in which the FFT size is fixed to maximize the speed. This plan/handle should be reused for rings of the same size until all rings of the same size are done.
4. Since the FFTs are done ring by ring for all maps, in the GPU implementation, once an FFT on one ring is done, the memory copy from the output of the FFT to the device memory can be done asynchronously with the unfinished FFT operations, which are performed on the CPU only. This overlaps the memory copy with the FFT operations and saves time significantly.
5. By default, the primary dimension of the FFT result is occupied by $m$. However, because $m$ does not participate in matrix multiplication, and current “gemm” programs requires continuous primary dimension; a transpose procedure is needed, which should be done by the FFT program via setting of the FFT I/O strides.
6. The most important point is that, the FFT-to-$a_{\ell m}$ operations (eqs. (3 & 5)) should be done by the matrix multiplication, which is a part of lower level BLAS libraries. These libraries are the core parts of numerous HPC applications, and therefore are usually fully supported and highly optimized. BLAS implementations, such as the Intel MKL library, the cuBLAS BLAS library, and third party libraries like Eigen, OpenBLAS and cUBLAS, have been widely used. Moreover, these implementations also allow various deployments, from laptop to super computers, from Windows to Linux, and from CPU platforms to GPU platforms. In this study, we use the “dgemm” subroutine from the Intel MKL library.

3 The “primary dimension” is the dimension with continuous storage. For example, in FORTRAN this is the first dimension, and in C this is the last dimension.
MKL BLAS library for the CPU code and the “cublas-Dgemm” subroutine from the NVIDIA cuBLAS library for the GPU code. These subroutines and their sisters\(^4\) have been proven to be the top-notch BLAS libraries.

7. To fully exploit the power of the “dgemm” or “cublas-Dgemm” implementation, the storage scheme should be carefully designed. We employ a novel storage scheme of \(\alpha_{\ell m s}\), which compactifies complex data into real matrices (see appendix A for details). Moreover, the built-in read-write support of the “dgemm” should be used as much as possible to reduce redundant read-write operations. In other words, the reading, conversion, computing, writing, and scaling are done by the “dgemm” as much as possible, and different types of the SHTs are implemented by changing the constants and the order of calling the “dgemm” subroutine.

8. When running with some iterative schemes, e.g., by setting a non-zero \(N_{\text{iter}}\) parameter in HEALPix, it is important to restrict the iterations in between \(T^{k}_{m,0}\) and \(a^{k}_{\ell m}\) (eq. (3) and eq. (5)), because the forward and backward FFT transforms between \(T^{k}_{m}(\theta)\) and \(T^{k}_{\ell m}(\theta, \phi)\) are lossless. Meanwhile, the iterative updating of \(a^{k}_{\ell m}\) should be done by the “dgemm” via an optimal setting of parameters to save time.

9. The program should support forward and backward transforms of \(a^{E}_{\ell m}\) and \(a^{B}_{\ell m}\) separately. In some data processing pipelines, each step may only need one of them (Liu et al. 2019; Liu et al. 2019; Liu 2019), so the speed of SHTs can be increased by 100% for free.

10. The ring optimization (Schaeffer 2013) should be used, which means for a given \(m\), the rings with extremely low amplitudes of \(P_{\ell m}\) are ignored. This saves about 15% of time, which is consistent with (Schaeffer 2013). Note that the ring optimization affects not only \(P_{\ell m}\), but also the FFT-mapping.

11. When \(N_{\text{side}} > 512\), it is necessary to scale some \(P_{\ell m}\)s during recursions, because the recursions start from the diagonal elements (\(\ell = m\)) that contain factor \(\sin^{m}(\theta)\), which can cause arithmetic underflow for a 64-bit floating number at high \(m\). However, a normal scaling method causes considerable performance loss when \(P_{\ell m}\) is required to be a matrix. Therefore, the following dedicated methods should be adopted: for \(N_{\text{side}} \leq 512\), we do not use the scaling; and for higher resolutions, the recursion starts from “safe positions” that allow the scaling to be ignored. Here the “safe positions” mean the \(\ell\)-positions from which the amplitudes of \(P_{\ell m}[\cos(\theta)]\) are larger than a given threshold, such as \(10^{-20}\). These positions are computed once and saved as an input table \(\ell_{\text{safe}}(m, \theta)\).

12. For the polarized SHT that deals with both E and B-modes (QU-to-EB), there are two possible orders of computation: 1) firstly QU-to-E and then QU-to-B; 2) firstly Q-to-EB and then U-to-EB. The second one is adopted because it helps to reduce the number of FFT-mapping operations by 50% and saves the memory compared to the first one.

### 3.2. Memory cost

In Table 1, we show the peak memory cost of performing SHTs on 1000 maps with \(N_{\text{side}} = 128\) for both the CPU and GPU implementations. Input maps, output spherical harmonic coefficients \((\alpha_{\ell m s})\), and all necessary buffers are included in the estimation. Note that the memory cost approximately scales quadratically with \(N_{\text{side}}\), and linearly with the number of maps; thus, Table 1 provides a baseline for a quick estimation of the peak memory cost for other cases. Generally speaking, to achieve the best performance, the memory cost of the workflow should be close to the physical memory capacity, especially for fastSHT-GPU.

| Polarization | Iteration | CPU memory | GPU memory |
|--------------|-----------|------------|------------|
| No           | No        | 4.4 (4.9) GB | 3.2 GB     |
| No           | Yes       | 5.9 (4.9) GB | 4.7 GB     |
| Yes          | No        | 8.8 (7.2) GB | 5.9 GB     |
| Yes          | Yes       | 11.8 (7.2) GB | 8.9 GB     |

Table 1. The peak memory cost of performing SHTs on 1000 maps with \(N_{\text{side}} = 128\). Input maps, output spherical harmonic coefficients \((\alpha_{\ell m s})\), and all necessary buffers are included in the estimation. The numbers in brackets are the CPU memory costs of the GPU implementation. Due to allocating pinned memory for buffer arrays, fastSHT-GPU consumes slightly more CPU memory in row 1.

### 3.3. Performance

We test the performance of the fastSHT code on a docker virtualized machine with state-of-the-art hardware — it contains an Intel Platinum 8336C CPU and an NVIDIA Tesla A100 GPU. We are limited to using 24 CPU cores since the performance of the CPU version is also limited by other hard-
The compilers used in these tests are the Intel FORTRAN compiler and the NVIDIA PGI (Portland Group Inc.) FORTRAN compiler for the CPU and GPU versions, respectively. Even though GPUs are known to be more powerful at single-precision floating-point (FP32) computations than double-precision ones (FP64), we stick to computing with double precision in all our tests to fully meet the precision requirement of various cosmological simulations and observations. The versions of compilers and software used in this article to generate the benchmarks are: Intel One API (2022.0.2), NVIDIA HPC SDK (22.3), Healpy (15.2 with default installation), fastSHT (Chi Tian et al. (2022) with default installation).

In our tests, we notice that the fastSHT-GPU usually introduces non-negligible initialization overhead. This overhead includes the initialization of the cuBLAS library and the pre-allocation of cache arrays. Moreover, when a GPU is involved, the CPU buffer arrays are allocated in the form of pinned arrays, which ensures faster communications with the GPU memory at the cost of a larger allocation overhead. Fortunately, the fastSHT-GPU only needs to be initialized once for any amount of computation in similar batches. For example, an extremely large number of maps can be packed into several smaller batches, and the fastSHT-GPU initialization overhead will only happen once for the first batch, and then all follow-up batches are free from this overhead. This feature makes this initialization overhead distinct from other in-computation costs, such as the data transfer cost between the CPU and GPU. On the other hand, for a complicated analysis that involves massive floating point calculation, the initialization overhead is relatively negligible (see Table 2 for a realistic example). Therefore, in all of the performance evaluations presented in this work, we choose to separate the time costs of the initialization overhead and main SHT computations, and exclude this overhead when quantifying the acceleration factor for the fastSHT-GPU code.

In Figure 1, we compare the time costs of fastSHT with Healpy for the non-polarized cases with/without iterations. The input map resolution is \( N_{\text{side}} = 128 \) and \( \ell_{\text{max}} = 3N_{\text{side}} - 1 \). As explained above, the fastSHT-GPU time cost includes the CPU-GPU memory exchange time, but excludes the initialization overhead. In the upper panel of Figure 1, we focus on the case without iteration: fastSHT-CPU gives acceleration factors between 2.0\( \times \) and 3.7\( \times \), increasing with \( N_{\text{maps}} \); whereas fastSHT-GPU gives acceleration factors close to 10\( \times \). When a standard iteration scheme is performed to both fastSHT and Healpy to improve the SHT accuracy, the advantage of fastSHT is more manifest because the initialization overhead happens only once, and the time cost of memory copying is also relatively smaller, as shown in the lower panel of Figure 1. An acceleration factor bigger than 5.0\( \times \) can be achieved for fastSHT-CPU with more than 4000 maps, whereas the acceleration factor of fastSHT-GPU reaches about 30\( \times \) at 8000 maps.

In Figure 2, we show how the performance of fastSHT changes with the map resolution \( N_{\text{side}} \). The panels from top to bottom show the results of \( N_{\text{maps}} = 4000, 1000, 250 \) respectively. The number of maps is different for different panels because we are limited by the GPU memory capacity. Due to better parallel efficiencies of Healpy at higher resolutions, the acceleration factor of fastSHT-CPU is gradually suppressed at higher resolutions (but still \( >1.6 \) in all cases). The fastSHT-GPU, on the contrary, exhibits an increasing acceleration factor with \( N_{\text{side}} \), which is up to 34\( \times \)
at higher resolutions. The exact performance improvement remains to be checked by a future hardware with larger GPU memory capacity. However, the tendency is evident: according to Figures 1–2, more GPU memory usage leads to higher performance; thus, using fastSHT-GPU is recommended for larger number of maps; meanwhile, if the number of map is fixed, then a higher resolution leads to a higher performance, until one hits the GPU memory limit.

In particular, to demonstrate potential applications of our new SHT scheme in a realistic situation, we perform a pipeline of fixing the EB-leakages by the best blind estimator (Liu et al. 2019; Liu 2019), which not only includes all types of SHTs (forward, backward, temperature, polarization, iterative scheme), but also involves a linear regression procedure and a lot of memory updating operations. We choose to work with 3 iterations and at two different resolutions: the first one is $N_{\text{side}} = 256$ and $N_{\text{maps}} = 2000$, and the second one is $N_{\text{side}} = 512$ and $N_{\text{maps}} = 500$. The number of CPU cores is still 24, and the time costs are listed in Table 2, which shows CPU acceleration factors of 4.5 × $N_{\text{side}}$ and 3.2 × $N_{\text{side}}$ and GPU acceleration factors of 13 × $N_{\text{side}}$ and 14 × $N_{\text{side}}$, respectively for the two tests. It is worth noting that the GPU initialization overheads in these two tests are less than 5 s, accounting for only 5–7% of the total time. This is a direct consequence of the massive floating-point computation required by the pipeline, and the fastSHT-GPU outperforms the traditional scheme significantly even after including the overhead, suggesting a great application potential of our SHT scheme in a computationally intensive CMB data processing.

| $N_{\text{side}}$ | $N_{\text{maps}}$ | $t_1$  | $t_2$  | $t_3$  | Overhead |
|-----------------|-----------------|-------|-------|-------|----------|
| 256             | 2000            | 815 s | 181 s (4.5 ×) | 62 s (13 ×) | 4.8 s     |
| 512             | 500             | 1325 s| 416 s (3.2 ×) | 95 s (14 ×) | 4.9 s     |

Table 2. The time costs for Healpy ($t_1$), fastSHT-CPU ($t_2$), and fastSHT-GPU ($t_3$) in a pipeline that fixes the EB-leakage. The CPU part is done with 24 cores. The GPU overhead is presented in the last column, and the acceleration factors relative to Healpy are given in brackets.

Lastly, we notice that Libsharp also supports an optimization option when processing many input maps based on a straightforward idea: compute the $P_{tm}$s only once, and then reuse them for each input map. Nevertheless, the calculation of the $P_{tm}$s only takes a small fraction of the total computational time in both Libsharp and fastSHT, and indeed we see no significant improvement when enabling this feature in Libsharp in our tests. Meanwhile, this is an advanced feature and is not included in Healpy. As a result, we ignore this feature in the Libsharp and do not cache $P_{tm}$s in our fastSHT code.

4. AN IMPROVED ITERATION SCHEME

![Figure 2](image-url)
mediately within the $m$-loop. The pseudo codes with more details for these two schemes are described in Algorithm 2 and 3 respectively.

**Algorithm 2:** Fast SHT algorithm for many sky maps with the traditional iterative method

**Data:** Input maps as matrix: $S$

Number of iterations: $N_{iter}$

**Result:** Spherical harmonic coefficients: $a_{\ell m}$

1. $F_{\ell m}^h = \text{Batch FFT}(S)$
2. for $n = 1$ to $N_{iter}$ do
3.  for $m = 0$ to $\ell_{\text{max}}$ do
4.   for $\ell = 0$ to $\ell_{\text{max}}$ do
5.     compute Legendre polynomials $P_m(\ell, \theta)$
6.   end
7. end
8. $\delta(T^h_{\ell m}) = (T^k_{\ell m}) - (a^k_{\ell m} \cdot P^k_{\ell m})$
9. update $F_{\ell m}^h$ from $\delta(T^h_{\ell m})$
10. end
11. for $m = 0$ to $\ell_{\text{max}}$ do
12.  for $\theta = 0$ to $\ell_{\text{max}}$ do
13.   compute Legendre polynomials $P_m(\ell, \theta)$
14. end
15. end
16. \[
    (a^k_{\ell m}) = (P^k_{\ell m} \cdot T^k_{\ell m})
    \]
17. end
18. end

**Algorithm 3:** Fast SHT algorithm for many sky maps with the immediate iterative method

**Data:** Input maps as matrix: $S$

Number of iterations: $N_{iter}$

**Result:** Spherical harmonic coefficients: $a_{\ell m}$

1. $F_{\ell m}^h = \text{Batch FFT}(S)$
2. for $n = 1$ to $N_{iter}$ do
3.  for $m = 0$ to $\ell_{\text{max}}$ do
4.   for $\theta = 0$ to $\ell_{\text{max}}$ do
5.     compute Legendre polynomials $P_m(\ell, \theta)$
6.   end
7. end
8. $\delta(T^h_{\ell m}) = (T^k_{\ell m}) - (a^k_{\ell m} \cdot P^k_{\ell m})$
9. update $F_{\ell m}^h$ from $\delta(T^h_{\ell m})$
10. end
11. for $k = 0$ to $N_{\text{maps}}$ do
12.  remap $F_{\ell m}^h$ to $(T^k_{\ell m})$
13. end
14. \[
    (a^k_{\ell m}) = (P^k_{\ell m} \cdot T^k_{\ell m})
    \]
15. end
16. end

We then compare these two iterative schemes in Figure 3, where the immediate mode is apparently more accurate when the input maps are band-limited to $\ell_{\text{max}} < 3N_{\text{side}}$. We also observe that the traditional mode gives better convergence when the input maps have a stronger band-limit of $\ell_{\text{max}} \leq 2N_{\text{side}}$; however, in this case, the errors of both modes are negligible.

**Figure 3.** The pixel domain errors of iterative solutions as $T_{\text{input}} - T_{\text{output}}$, for the traditional mode (left) and the immediate mode (right) with 3 rounds of iterations. The input maps are CMB intensity-only simulations based on the Planck 2018 best-fit CMB spectrum (Planck Collaboration et al. (2018)), with $N_{\text{side}} = 128$ a band limit of $\ell_{\text{max}} \leq 383$.

5. SUMMARY AND DISCUSSIONS

In this work, we design a dedicated algorithm for SHTs on a large number of sky maps, which is also a complete SHT toolkit for the CMB data, including temperature, polarization; forward, backward; and iterative, non-iterative transforms. The results of the new code are identical to the traditional SHT toolkit like Healpy, but the performance is significantly improved on a large number of sky maps. With the benchmark environment described in section 3.3: For the CPU implementation, the speed of the new code is $2 \sim 10$ times of Healpy; when GPU is employed, the typical acceleration factor is more than one order of magnitude without iteration, and can be further boosted to about $30 \times$ with 3 rounds of iterations. By applying this new scheme to the standard fix-EB leakage pipeline for 2000 masked CMB maps at $N_{\text{side}} = 256$, even after including the initialization overhead, the GPU version still gets a $10 \times$ speed-up compared to the Healpy SHT toolkit that does the same job. Moreover, the acceleration factor of the GPU version increases almost linearly with $N_{\text{side}}$, indicating that the new code can be much more efficient at a higher resolution (given enough GPU memory). As far as we know, this is the fastest SHT implementation for a large number of spherical maps till the present day.

However, we would like to point out the following facts: 1) A dedicated compiler flag optimization can help to increase the performance of Healpy by roughly 100%; thus, in this case, the acceleration factors should be divide by two. 6 Note that if the input map has no band limit (like a white noise map), the situation is more complicated and needs a further study.
2) The performance of fastSHT is insensitive to the compiling flags, because the most time consuming part (matrix multiplication) is done by either the Intel-MKL or NVIDIA CUDA libraries, which are already optimized. 3) For a single input map, the computation of SHT will degenerate from the BLAS level 3 to level 2, so the core part of the new scheme will become less efficient. Moreover, the acceleration requires using $P_{\ell m}$s as matrices, but for a single map, it is more efficient to compute the $P_{\ell m}$s in small segments to use the CPU L1 cache more efficiently. Similar reasons apply for the GPU code, and the large initialization overheads also prevent the GPU code, and the large initialization overheads also prevent the fastSHT-GPU from outperforming Healpy when dealing with only a small number of high-resolution maps. 4) Since it is unlikely that one algorithm can be the best for all cases, and the performances are significantly affected by the hardware and software environments, we have provided a test script called “run_nproc.sh” to help readers run a batch of tests on their own machine to determine which one is more suitable in a user-specified environment. 5) We are also considering integrating the results of such test in the fastSHT to make an automatic choice of the SHT engine for various computer hardware and data scales.

We have also provided a new algorithm integrated in the fastSHT to improve the accuracy of SHTs by optimizing the iteration strategy. The test results show that this new iterative scheme is more accurate at high multiples, especially for the temperature maps. We also point out that when the input map is strictly band-limited to $\ell < 2N_{\text{side}}$, the traditional iterative scheme is slightly better – but both iterative schemes nevertheless give negligible errors in this case.

Numerous ongoing and planned CMB experiments are going to bring heavy data forecasting and analyzing tasks that usually involve processing large amounts of mock data. As the core component in almost all CMB data processing pipelines, the performance of SHTs has become critical. The new scheme introduced in this paper fully exploits the state-of-the-art software and hardware to accelerate SHTs. As a consequence, the time costs for SHT operations are reduced considerably, and standard data processing pipelines, such as fixing EB-leakages, can be accelerated correspondingly. Also note that SHTs are widely used not only in astrophysics and cosmology but in other areas of science as well, including geoscience, atmospheric sciences, oceanography, etc., where spherical data is used frequently. As a result, we anticipate that our new scheme will have broader applications.

Finally, we would like to refer to the most recent development of the multi-GPU computation technique that uses distributed GPU memory in a 2D block-cyclic fashion (The NVIDIA company 2022). This technique is still in an early view stage, and it is expected to provide access to much more GPU memory and computational power in a dedicated way for matrix multiplications; hence it will likely solve the GPU memory size problem that prevents our fastSHT-GPU code from working sufficiently well at a very high resolution, like $N_{\text{side}} \geq 2048$. We shall keep tracking the development of this technique and update our code accordingly in the future.

This work is supported by the National Key R&D Program of China (2021YFC2203100, 2021YFC2203104) and the Anhui project Z010118169. We would like to thank the referee’s professional comments that helped improve this work. We also thank Matthew Carney, James Mertens, and Glenn Starkman for helpful discussions, and our USTC colleagues for helping us with the GPU test environment.

APPENDIX

A. A FEW TECHNICAL TIPS

A.1. FFT-mapping with the unit circle

One thing to be noticed in a SHT is that, in most cases, the number of pixels in one ring and the value of $m$ do not match, thus one should properly map $m$ to the FFT frequency circle, as shown in Figure 4, which is a unit circle centered around the original point, with points on it marked by the azimuthal angles $\phi$. In the frequency circle, $\phi = 0$ correspond to the zero frequency, $\phi = \pi$ correspond to the Nyquist frequency, and all frequencies above the Nyquist frequency will be mapped to lower frequencies by moving along the circle in an anti-clockwise direction. This is also a clear presentation of the Nyquist-Shannon sampling theorem, which naturally shows why one cannot exceed the Nyquist frequency. As an example, $m = 0$ is always mapped to $\phi = 0$, regardless of the value of $N$ (size of one ring); but other $m$ should be mapped to $\phi = \frac{2\pi m}{N}$. This means, even if $m$ is the same, in rings with different $N$, it will be mapped to different positions, and vice versa.

A.2. Polynomials for polarization

In the context of rotation, $Q$ and $U$ can be decomposed into spin $\pm 2$ spherical harmonics Zaldarriaga & Seljak (1997); Zaldarriaga (1998) as follows:

$$Q(\hat{n}) \pm iU(\hat{n}) = \sum_{l,m} a_{\pm 2,lm} \pm 2 Y_{lm}(\hat{n}), \quad (A1)$$

where $\pm 2 Y_{lm}(\hat{n})$ are the spin $\pm 2$ weighted spherical harmonics. Thus, the spin $\pm 2$ spherical harmonic coefficients $a_{\pm 2,lm}$
are given by the following:

\[
a_{\pm 2,\ell m}^k = \sum_{\theta} P_{\pm 2,\ell m}(\cos(\theta))(Q_{\ell m}^k \pm i U_{\ell m}^k),
\]

where \(Q_{\ell m}^k\) and \(U_{\ell m}^k\) are the Fourier transforms of the Q and U Stokes parameters of the \(k\)-th map at the \(i\)-th ring. For simplicity, we define the following matrix forms:

\[
\mathbf{A}_m^\pm = (a_{\pm 2,\ell}^k)_{m,\ell}, \quad \mathbf{P}_m^\pm = (P_{\pm 2,\ell})_{m,\ell}
\]

Because the E- and B-mode spherical harmonic coefficients are defined as

\[
a_{E,\ell m} = -(a_{2,\ell m} + a_{-2,\ell m})/2, \
a_{B,\ell m} = i(a_{2,\ell m} - a_{-2,\ell m})/2,
\]

In this work, we use a HEALPix-like program to compute the \(P_{\ell m}\) coefficients for both temperature and polarization. One should note that (seems missing in the HEALPix manual), the convention is to compute the summation and difference of the polarized \(P_{\ell m}\) coefficients, which are used in SHTs instead of polarized \(P_{\ell m}\)s themselves. With such convention, the polarized \(a_{\ell m}\)S are computed from the Q and U Stokes parameters by

\[
-F_{\ell m}^+(\sigma) = \frac{1}{2} |Y_{+2,\ell m}(\sigma) + Y_{-2,\ell m}(\sigma)| = \lambda_{\ell m}^+(\theta)e^{i m \phi}
\]

\[
-F_{\ell m}^-(\sigma) = \frac{1}{2} |Y_{+2,\ell m}(\sigma) - Y_{-2,\ell m}(\sigma)| = \lambda_{\ell m}^-(\theta)e^{i m \phi},
\]

where \(Y_{\pm 2,\ell m}\) are the spin\(\pm 2\) spherical harmonics, and \(\lambda_{\ell m}^{\pm}(\theta)\) are the effective Legendre coefficients generated by HEALPix. Let \(X\) be either \(Q\) or \(U\), then the E and B-mode spherical harmonic coefficients, computed with the

**HEALPix convention**, are

\[
X_{\ell m}^\pm = \int X(\sigma) F_{\ell m}^{\pm}(\sigma) d\sigma
\]

\[
a_{E,\ell m} = -Q_{\ell m}^+ - i U_{\ell m}^-
\]

\[
a_{B,\ell m} = -U_{\ell m}^+ - i Q_{\ell m}^-
\]

Lastly, a new storage scheme of \(a_{\ell m}\) is introduced to match the requirement of high performance computing with many input maps, in which all \(a_{\ell m}\) coefficients are stored in a real-valued square matrix of size \((\ell_{\text{max}} + 1) \times (\ell_{\text{max}} + 1)\). The real and imaginary parts are compactly\(^7\) stored in the square matrix with a very simple addressing rule (see also Figure 4):

Real part: \((\ell, m) \rightarrow (\ell, m)\)

Imaginary part: \((\ell, m) \rightarrow (\ell - m, \ell_{\text{max}} - m + 1),\)

where we have the natural constraints that all non-zero imaginary parts satisfy: \(1 \leq m \leq \ell_{\text{max}}\) and \(m \leq \ell \leq \ell_{\text{max}}\). This new scheme is especially important for the GPU-implementation, because this kind of regular shape tensor-style storage with very simple addressing rule is apparently favored by the GPU. Even in a CPU-implementation, it is able to significantly reduce the time cost of memory operations.

**B. THE POSSIBILITY OF ALLOWING MORE ITERATIONS**

We know that the accuracy of SHTs can be improved with an increasing number of iteration rounds. The default choice in Healpy is 3 rounds, which is a balance between the time cost and accuracy – with the “time cost” estimated under the old scheme (The HEALPix team 2005). However, fastSHT-GPU is significantly more efficient with a bigger number of iterations, because the iterative part of SHT is done in GPU instead of CPU. This fact will break the old balance and allow more iterations at a reasonable time cost. In order to evaluate this effect, We have done a non-polarization fastSHT-GPU test with \(N_{\text{side}} = 256\), \(\ell_{\text{max}} = 3N_{\text{side}} - 1\), \(N_{\text{maps}} = 1000\) and \(N_{\text{iter}} = 3\) and 9 respectively: When \(N_{\text{iter}} = 3\), the time cost of the non-iterative part (the operations outside the iteration) is and each round of iteration (including two SHTs, one forward and one backward) costs only 0.444 s. Thus we can predict the time cost with any number of iterations by the following equation:

\[t_{\text{total}} = t_0 + (2N_{\text{iter}} + 1)t_1,\]

and for the example here, we have \(t_0 = 0.592\) s and \(t_1 = 0.222\) s respectively. To double check the above equation: for

\(^7\) Here “compact” means all matrix elements are used without redundancy.
\( N_{\text{iter}} = 9 \), the total time cost is 4.800 s, which is consistent with the time computed from the above equation. Therefore, when \( N_{\text{iter}} \) increase from 3 to 9, the time cost only increases from 2.1 s to 4.8 s, indicating that we get a relatively higher efficiency when \( N_{\text{iter}} \) increases for fastSHT-GPU.

REFERENCES

Abazajian, K. N., Adshead, P., Ahmed, Z., et al. 2016, ArXiv e-prints. https://arxiv.org/abs/1610.02743

Ade, P., Aguirre, J., Ahmed, Z., et al. 2019, JCAP, 2019, 056, doi: 10.1088/1475-7516/2019/02/056

Austermann, J. E., Aird, K. A., Beall, J. A., et al. 2012, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 8452, Millimeter, Submillimeter, and Far-Infrared Detectors and Instrumentation for Astronomy VI, ed. W. S. Holland & J. Zmuidzinas, 84521E, doi: 10.1117/12.927286

Chi Tian, Siyu Li, mreineck, & Hao Liu. 2022, fastSHT – the frozen GitHub repository for publication, Zenodo, doi: 10.5281/zenodo.7084233

Doroshkevich, A. G., Naselsky, P. D., Verkhodanov, O. V., et al. 2005, International Journal of Modern Physics D, 14, 275, doi: 10.1142/S0218271805006183

Fabbian, G., Szydlarski, M., Stompor, R., Grigori, L., & Falcou, J. 2012, Publications of the Astronomical Society of the Pacific, ASP Conference Series, 61

Górski, K. M., Hivon, E., Banday, A. J., et al. 2005, ApJ, 622, 759, doi: 10.1086/427976

Goto, K., & Van De Geijr, R. 2008, ACM Trans. Math. Softw., 35, doi: 10.1145/1377603.1377607

Hazumi, M., Borrill, J., Chinone, Y., et al. 2012, in Proc. SPIE, Vol. 8442, Space Telescopes and Instrumentation 2012: Optical, Infrared, and Millimeter Wave, 844219, doi: 10.1117/12.926743

Huffenberger, K. M., & Wandelt, B. D. 2010, The Astrophysical Journal Supplement Series, 189, 255, doi: 10.1088/0067-0049/189/2/255

Hupca, I. O., Falcou, J., Grigori, L., & Stompor, R. 2012, in Euro-Par 2011: Parallel Processing Workshops (Berlin, Heidelberg: Springer Berlin Heidelberg), 355–366

Keating, B., Moyerman, S., Boettger, D., et al. 2011, ArXiv e-prints. https://arxiv.org/abs/1100.2101

Keating, B. G., Ade, P. A. R., Bock, J. J., et al. 2003, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 4843, Polarimetry in Astronomy, ed. S. Fineschi, 284–295, doi: 10.1117/12.459274

Kermode, J. R. 2020, Journal of Physics: Condensed Matter, 32, 305901, doi: 10.1088/1361-648x/ab82d2

Li, H., Li, S.-Y., Liu, Y., et al. 2018, National Science Review, 6, 145, doi: 10.1093/nsr/nwy019

Liu, H. 2019, Journal of Cosmology and Astroparticle Physics, 2019, 001, doi: 10.1088/1475-7516/2019/10/001

Liu, H., Creswell, J., & Dachlythra, K. 2019, Journal of Cosmology and Astroparticle Physics, 2019, 046, doi: 10.1088/1475-7516/2019/04/046

Liu, H., Creswell, J., von Hausegger, S., & Naselsky, P. 2019, PhRvD, 100, 023538, doi: 10.1103/PhysRevD.100.023538

McEwen, J. D., Puy, G., Thiran, J. P., et al. 2013, IEEE Transactions on Image Processing, 22, 2275, doi: 10.1109/TIP.2013.2249079

Mohlenkamp, M. J. 1999, Journal of Fourier Analysis and Applications, 5, 159, doi: 10.1007/BF01261607

Niemack, M. D., Ade, P. A. R., Aguirre, J., et al. 2010, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 7741, Millimeter, Submillimeter, and Far-Infrared Detectors and Instrumentation for Astronomy V, ed. W. S. Holland & J. Zmuidzinas, 77411S, doi: 10.1117/12.857464

Planck Collaboration, Aghanim, N., Akrami, Y., et al. 2018, arXiv e-prints, arXiv:1807.06209. https://arxiv.org/abs/1807.06209

Reinecke, M. 2011, A&A, 526, A108, doi: 10.1051/0004-6361/201015906

Reinecke, M., & Seljebotn, D. S. 2013, A&A, 554, A112, doi: 10.1051/0004-6361/201321494

Rubio-Martín, J. A., Rebolo, R., Tucci, M., et al. 2010, in Astrophysics and Space Science Proceedings, Vol. 14, Highlights of Spanish Astrophysics V, 127, doi: 10.1007/978-3-642-11250-8_12

Schaeffer, N. 2013, Geochemistry, Geophysics, Geosystems, 14, 751, doi: 10.1002/bgge.20071

Suda, R., & Takami, M. 2002, Math. Comput., 71, 703, doi: 10.1090/S0025-5718-01-01386-2

The HEALPix team. 2005, online document, https://healpix.sourceforge.io/html/sub_map2alm_iterative.htm

The NVIDIA company. 2022, The cuBLAS preview page, https://developer.nvidia.com/cublas

The Planck Collaboration. 2006, arXiv e-prints, astro. https://arxiv.org/abs/astro-ph/0604069

Zaldarriaga, M. 1998, The Astrophysical Journal, 503, 1. http://stacks.iop.org/0004-637X/503/i=1/a=1

Zaldarriaga, M., & Seljak, U. c. v. 1997, Phys. Rev. D, 55, 1830, doi: 10.1103/PhysRevD.55.1830