applied sciences

Article

Research on High-Performance Fourier Transform Algorithms Based on the NPU

Qing Li [1,2], Decheng Zuo [1], Yi Feng [1] and Dongxin Wen [1,*]

[1] Faculty of Computing, Harbin Institute of Technology, Harbin 150001, China; 21b903107@stu.hit.edu.cn (Q.L.); zuodc@hit.edu.cn (D.Z.); fengy@ftcl.hit.edu.cn (Y.F.)
[2] Jiangsu Automation Research Institute, Lianyungang 222006, China
* Correspondence: wdongxin@hit.edu.cn

Abstract: Backpack computers require powerful, intelligent computing capabilities for field wearables while taking energy consumption into careful consideration. A recommended solution for this demand is the CPU + NPU-based SoC. In many wearable intelligence applications, the Fourier Transform is an essential, computationally intensive preprocessing task. However, due to the unique structure of the NPU, the conventional Fourier Transform algorithms cannot be applied directly to it. This paper proposes two NPU-accelerated Fourier Transform algorithms that leverage the unique hardware structure of the NPU and provides three implementations of those algorithms, namely MM-2DFT, MV-2FFTm, and MV-2FFTv. Then, we benchmarked the speed and energy efficiency of our algorithms for the gray image edge filtering task on the Huawei Atlas200I-DK-A2 development kits against the Cooley-Tukey algorithm running on CPU and GPU platforms. The experiment results reveal MM-2DFT outperforms OpenCL-based FFT on NVIDIA Tegra X2 GPU for small input sizes, with a 4- to 8-time speedup. As the input image resolution exceeds 2048, MV-2FFTv approaches GPU computation speed. Additionally, two scenarios were tested and analyzed for energy efficiency, revealing that cube units of the NPU are more energy efficient. The vector and CPU units are better suited for sparse matrix multiplication and small-scale inputs, respectively.

Keywords: backpack computer; wearables; NPU; Fourier Transform

check for updates

Citation: Li, Q.; Zuo, D.; Feng, Y.; Wen, D. Research on High-Performance Fourier Transform Algorithms Based on the NPU. *Appl. Sci.* **2024**, *14*, 405. https://doi.org/10.3390/app14010405

Academic Editor: Paris Kitsos

Received: 11 November 2023
Revised: 22 December 2023
Accepted: 29 December 2023
Published: 1 January 2024

1. Introduction

Smart wearable devices are playing an increasingly important role in the application scenario of field special operations. Among these devices, the backpack computer serves as the "central nervous system" for edge intelligence in the wearable device group. It collects a variety of measurement data from multiple devices, such as smartwatches, glasses, belts, helmets, and wristbands, for centralized processing. This organizational approach follows the edge computing paradigm [1,2]. Consequently, the backpack computer requires high computing power to accelerate scientific calculations or intelligent inference. One common strategy to meet that need is the incorporation of additional hardware, such as graphics processing units (GPUs) or field programmable gate arrays (FPGAs). However, due to the limited size and battery capacity of backpack computers, adding extra hardware leads to adverse effects, including increased heat generation and reduced battery life.

The system-on-chip (SoC) with high sampling performance emerges as the optimal solution that effectively balances performance and power consumption. For intelligent wearable applications, however, most on-chip GPUs currently exhibit poor compatibility with mainstream intelligent computing frameworks. Additionally, incorporating an external FPGA for intelligent inference is only suitable for specialized applications and presents challenges in development. Fortunately, SoC-containing NPUs have been developed rapidly in recent years, driven by the flourishing of smartphones [3,4]. In intelligent applications, NPUs boost the computational speed of matrix and vector operations through data parallelism [5,6]. This not only substantially accelerates intelligent inference but also enhances energy efficiency. Therefore, for most intelligent computing scenarios, the CPU + NPU architecture is sufficient and does not require additional heterogeneous computing hardware units [7–10].

The computational demands for intelligent applications may not always be adequately addressed in the calculation phase of intelligent inference, as numerous data preprocessing tasks often require high computing power as well. In various application scenarios of wearable devices, such as heart rate estimation [11] and human activity recognition [12], it is necessary to apply the Fourier Transform (FT) to convert sensor sampling data from the time-domain signals into the frequency-domain signals before performing intelligent inference. Moreover, the two-dimensional Fourier Transform has been widely used in edge intelligence scenarios such as road object detection [13] and stroke detection [14]. With the advancement in signal sampling rate and resolution, the computational consumption of the FT has notably increased, especially for image Fourier Transforms. For a backpack computer with CPU + NPU architecture, executing two-dimensional or high-dimensional Fourier Transforms with CPUs can result in declines in real-time signal processing performance, potentially leading to performance bottlenecks in computing. Therefore, it is crucial to prioritize the research of accelerating FT on NPUs.

Migrating existing FT algorithms directly to NPUs is infeasible due to the distinctive hardware architecture of NPUs. This is because the matrix and vector units of NPUs process data blocks in a serial manner, while GPUs and FPGAs process bytes with multiple threads in parallel [15]. Consequently, the conventional fast FT algorithm may not be suitable for NPUs. To address this issue, this paper investigates the design of acceleration algorithms for Fourier Transform utilizing NPU's distinctive architecture. It aims to explore the applicability of NPU in typical algorithms that preprocess data for AI and to identify methods for enhancing NPU efficiency. Two novel NPU algorithms are designed and implemented: the direct matrix multiplication-based DFT algorithm (MM-2DFT) and the matrix-vector iterative operation-based Fast Fourier Transform (FFT) algorithm (MV-2FFT). In benchmarking against the GPU OpenCL-based Cooley-Tukey algorithm running on NVIDIA Tegra X2, our MM-2DFT algorithm outperforms on small inputs, while the MV-2FFT algorithm shows superior performance on larger inputs. Based on these algorithms, we then discussed the energy efficiency of various computing units within the NPU. In summary, our contributions are as follows:

- We analyzed the Discrete Fourier Transform (DFT) and several typical Fast Fourier Transform (FFT) algorithms. Then, we analyzed the adaptability of those algorithms and designed corresponding acceleration strategies based on the distinctive hardware architecture of NPU;
- We have presented the direct matrix multiplication algorithm (MM-2DFT), which can fully utilize the computing power of the cube unit for accelerating the DFT algorithm. The MM-2DFT is highly suitable for scenarios with smaller inputs;
- We have presented the matrix-vector iterative operation-based FFT algorithm (MV-2FFT), which uses both the cube and vector unit to perform the FFT algorithm based on the matrix divide-and-conquer method. Building on this algorithm, we developed two implementations to evaluate the performance difference between matrix and vector multiplication on various computation units of NPUs. The results revealed the fact that (1) the MV-2FFT is better suited for scenarios with larger inputs, and (2) vector multiplication is quicker and more energy-efficient than matrix multiplication on NPUs;
- We deployed the NPU algorithm implementations on a real hardware platform and evaluated their performance and power consumption in a typical image preprocessing task. Based on that evaluation, we comprehensively analyzed the energy efficiency of different algorithms and the typical NPU's internal hardware units.

2. Related Works

2.1. The Applications and Acceleration Research of the Fourier Transform

2.1.1. The Applications of Fourier Transform

The Fourier Transform is a widely adopted spectral analysis method in the preprocessing stage of intelligent computing applications. These applications span various domains, including image recognition, fatigue detection, material design, and more. In detail, Balabanova et al. [16] proposed a feature extraction approach that involves preprocessing images in various graphic formats using FFT and spectral analysis methods. Sedik et al. [17] proposed a fatigue detection system that utilizes FFT for feature extraction and noise elimination. Seyed Mahmoud et al. [18,19] calculated the thermal conductivity based on the microstructure geometry of the provided material using an FFT-based method. Subsequently, they developed a machine learning method to design materials according to desired performance.

2.1.2. The Research on Accelerating the Fourier Transform

Due to the inherent computational complexity of the Fourier Transform algorithm, specialized computational acceleration is traditionally required, particularly through the utilization of heterogeneous computing architectures in various engineering applications. Heterogeneous acceleration of the Fourier Transform and its fast algorithms have been extensively studied, but most of these studies are based on specialized hardware for the Fourier Transform, such as GPU processors or FPGA. For instance, Le Ba et al. [20] proposed a new FFT hardware that enhanced throughput by simplifying computation and reducing memory usage. Their architecture also reduces energy consumption by improving input scheduling algorithms. Zhao et al. [21] proposed a new large-scale FFT framework that improves parallelism efficiency by reducing communication overhead using advanced floating-point compression techniques. Ayala et al. [22] studied multidimensional FFT on large-scale GPU systems. They evaluated the features of parallel FFTs, optimized performance with tuning methodology, and achieved linear scalability on these GPU systems.

2.2. NPU Research for Edge Devices

Due to superior computing power and energy efficiency, the utilization of NPUs for accelerating intelligent computing on mobile edge devices has become a hot research area. Current research in this field mainly focuses on (1) enhancing inference accuracy and (2) boosting NPU efficiency.

Regarding enhancing inference accuracy, Tan et al. [3,4] focused on optimizing the trade-off between inference time and accuracy within specific constraints. They also addressed the distribution of workload between the CPU and NPU. Lee et al. [23,24] concentrated on improving the performance and accuracy of inference by identifying optimal model structures using model search methods. They also developed an optimization framework to generate optimal NPU dataflow for more accurate and faster deep learning models.

Regarding boosting the inference efficiency of NPUs, Rapp et al. [25] explored ways to enhance the inference efficiency of the NPU by examining the dynamic frequency characteristics of the processor. This was achieved by managing the temperature of the processor cluster with task migration and Dynamic Voltage and Frequency Scaling (DVFS) techniques. Xue et al. [6] introduced a novel framework for managing NPU resources, enhancing efficiency and fairness in multi-user scenarios by enabling concurrent execution, and resource sharing. In addition to these works, there are also studies focused on improving NPU inference efficiency by accelerating parameter updating [26] or task scheduling [27,28].

2.3. Summary

In conclusion, the Fourier Transform and its variants, as widely used data preprocessing techniques, find extensive applications in various scenarios (Section 2.1.1). Numerous research studies have explored heterogeneous acceleration techniques for these transforms

(Section 2.1.2). However, the predominant focus of these studies has been on GPUs and FPGAs, with comparatively less attention given to NPUs. This trend can be attributed to the fact that although the NPU primarily serves as an accelerator for deep neural networks, its architecture, particularly in data parallel processing, makes it suitable for computationally intensive tasks. This suitability arises from NPUs typically incorporating dedicated hardware units designed for matrix and vector calculations, which enhance the efficiency and parallelism of these calculations through specialized instructions. However, ongoing research and development efforts in NPUs mainly focus on accelerating neural networks, and support for typical engineering algorithm computation remains insufficient. Therefore, employing NPUs to accelerate typical engineering algorithms, especially in the study of the Fourier Transform, is significant.

3. Background

To facilitate our study, we introduce the NPU architecture and analyze the characteristics of the Fourier Transform and its fast algorithms.

3.1. Architecture of a Typical NPU

The typical NPU architecture is equipped with specialized hardware units that can directly perform matrix or vector multiplication and addition to accelerate neural network computations [6]. Take the Huawei Atlas200 AI accelerator module as an example; its built-in AI Core adopts the DaVinci architecture, as shown in Figure 1 [29].

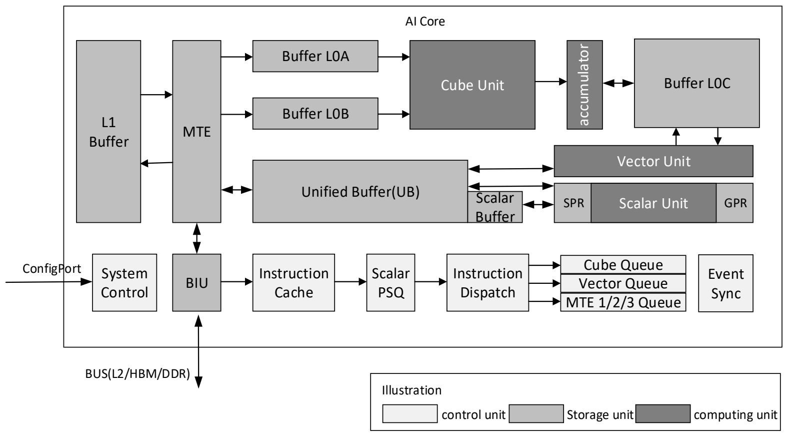

Figure 1. Huawei DaVinci architecture.

The AI Core is the primary hardware for computational power in NPUs, which contains a cube unit, vector unit, and scalar unit. The scalar unit is responsible for various scalar data operations and handles logical tasks for loop control and branch decisions in operators. The cube unit and vector unit are well-suited for computationally intense tasks involving matrix and vector operations. Specifically, each execution of the cube unit completes matrix multiplication on two 16×16 matrices of the float16 type. Each execution of the vector unit can simultaneously perform operations such as multiplication, addition, or subtraction on two 128-length vectors of the float16 type.

3.2. The Matrix Acceleration Technique in the Discrete Fourier Transform (DFT)

The Fourier Transform is a mathematical procedure that extracts frequency components from input data samples. That transform is heavily used in signal processing algorithms, such as equalization, filtering, compression, and more.

For frequency domain analysis, the DFT is frequently used in engineering, and its Formula is shown as follows:

$$X_j = \sum_{k=0}^{n-1} x_k \exp(-\frac{2\pi i}{n} jk) \;\; j = 0, 1, \ldots, n-1 \tag{1}$$

The multiplication of an N-by-N DFT matrix and the discrete samples of the sequence of length N can be expressed in the form of Formula (2), where $\omega = e^{\frac{-2\pi}{N} i}$. The complexity of the algorithm reaches $O(N^2)$.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ \cdots \\ x[N-2] \\ x[N-1] \end{bmatrix} = \begin{bmatrix} y[0] \\ y[1] \\ \cdots \\ y[N-2] \\ y[N-1] \end{bmatrix} \tag{2}$$

From Formula (2), it can be seen that the DFT is the process of multiplying a DFT matrix of $N \times N$ with an input vector of $N \times 1$; this task is well suited to the powerful matrix processing capabilities of NPUs. When the input vector length is $N \leq 16$, the DFT process can be executed as a single matrix multiplication instruction. When $N > 16$, the matrix multiplication can be decomposed into multiple low-order matrix multiplications using the block matrix principle, allowing the cube unit to handle the computation. Consequently, the computational complexity is significantly reduced to 1/256 of the original complexity. The direct calculation method still retains its relatively high-level calculation process, which means that as the order of matrices involved in matrix multiplication increases, the processing speed significantly decreases. However, that method holds engineering significance for lower-order inputs. This is due to the NPU's exceptional matrix computing capabilities and the finite nature of sample length N in typical engineering applications. Therefore, employing an NPU for direct DFT calculations using Formula (2) is a valuable engineering approach that effectively utilizes the computing power of the NPU cube unit.

3.3. The Acceleration Technique in the FFT Based on the Cooley-Tukey Algorithm

Due to the high computational complexity of solving Formula (2) directly, in engineering the Fast Fourier Transform (FFT) method is generally used. The FFT is a general term for techniques used to calculate DFT efficiently and quickly on computers. The most classic method in the FFT is the Cooley-Tukey algorithm [30]. The Cooley-Tukey algorithm reduces the complexity to O ($N \log_2 N$) by employing a divide-and-conquer strategy. The core idea of the Cooley-Tukey algorithm is to decompose an input vector of length N into two input vectors of length $N/2$ and recursively compute their Fourier Transforms with a similar method. The transformed input vectors are then combined to obtain the Fourier Transform result of the original input vector of length n. This combination process is achieved using butterfly operations. By repeatedly applying this decomposition and combination process, the Cooley-Tukey algorithm efficiently calculates the Fourier Transform with less computation. The acceleration performance of the Cooley-Tukey algorithm has been validated through numerous instances on both CPUs and GPUs.

However, implementing the Cooley-Tukey algorithm directly on NPUs poses a significant challenge due to the NPUs' inability to manipulate fine-grained data blocks, which is essential in the execution of that algorithm. This is because the Cooley-Tukey algorithm involves the butterfly operation. During the first several iteration cycles, the operation directly manipulates individual elements rather than data blocks consisting of at least 32 bytes, which is difficult for NPUs. In contrast, the major computing units of CPUs or GPUs can operate on a byte. Therefore, it is not suitable to utilize the NPU's vector units to accelerate the FFT based on the Cooley-Tukey algorithm.

3.4. The Acceleration Technique in the FFT Based on the Matrix Divide-and-Conquer Method

We were able to derive a matrix form of the Cooley-Tukey algorithm, presented in Formula (3), by incorporating the divide-and-conquer technique [31]. With that formula, we can control the granularity of matrix factorization within the feasible range of the NPU.

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix}_{N\times 1} = \begin{bmatrix} I & D \\ I & D \end{bmatrix} \times \begin{bmatrix} A \\ B \end{bmatrix}_{N\times 1} \tag{3}$$

where A and B are the result of multiplying the even and odd sequences of input data X by a DFT matrix of order $N/2$, respectively; $I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}_{\frac{N}{2}\times\frac{N}{2}}$, and D is the rotation matrix, where $D = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & \omega_N^1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \omega_N^{\frac{N}{2}-1} \end{bmatrix}_{\frac{N}{2}\times\frac{N}{2}}$, $X_1 = \begin{bmatrix} X(0) \\ X(1) \\ \vdots \\ X\left(\frac{N}{2}-1\right) \end{bmatrix}_{\frac{N}{2}\times 1}$, $X_2 = \begin{bmatrix} X\left(\frac{N}{2}\right) \\ X\left(\frac{N}{2}+1\right) \\ \vdots \\ X(N-1) \end{bmatrix}_{\frac{N}{2}\times 1}$.

Furthermore, we obtained Formula (4), which is used in the merging process:

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} A + DB \\ A - DB \end{bmatrix} \tag{4}$$

During the solving of A and B, we recursively repeat this process of decomposition and merging. In this way, the DFT transformation of a higher-dimensional vector is transformed into that of multiple lower-dimensional vectors. This algorithm has the same asymptotic time complexity as the Cooley-Tukey algorithm, but its calculation amount depends on the order of the vector yields by the last decomposition.

It can be observed from Formula (4) that this algorithm consists of matrix addition, subtraction, and multiplication operations and is appropriate for acceleration utilizing the cube units and vector units of NPUs in combination.

4. Methods

In wilderness formation tasks, it is common for backpack computers to preprocess images captured by wearable image sensors. This typically involves using the 2D Fourier Transform before applying intelligent processing to the images. The 2D Fourier Transform requires performing 1D Fourier Transforms on the rows and columns of the image individually. That means the calculation process comprises three steps, namely: (1) performing the Fourier Transformation according to Formula (5), (2) transposing the result obtained in step 1, and (3) performing another Fourier Transformation using Formula (5).

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \cdots & \omega^{(N-1)^2} \end{bmatrix} \times \begin{bmatrix} x[0] & x[1] & \cdots & x[N-1] \\ x[1] & x[2] & \cdots & x[2(N-1)] \\ \vdots & \vdots & & \vdots \\ x[N-1] & x[2(N-1)] & \cdots & x\left[(N-1)^2\right] \end{bmatrix} = \begin{bmatrix} y[0] & y[1] & \cdots & y[N-1] \\ y[1] & y[2] & \cdots & y[2(N-1)] \\ \vdots & \vdots & & \vdots \\ y[N-1] & y[2(N-1)] & \cdots & y\left[(N-1)^2\right] \end{bmatrix} \tag{5}$$

Considering that the forward and inverse transformations are commonly used in pairs in engineering applications, the inverse transformation algorithm adds only a conjugation operation based on the forward transformation algorithm, thus having little influence on the software architecture. Moreover, the inverse transform can intuitively demonstrate the effects of the forward transform, helping to validate the correctness of the transform algorithms. Therefore, the image preprocessing process is designed for both forward and inverse transformations.

In this section, two algorithms based on NPUs were designed to accelerate the application of the 2D Fourier Transform in image processing for formation task scenarios. One uses the direct matrix multiplication, while the other uses the divide-and-conquer method.

4.1. The Direct Matrix Multiplication-Based DFT Algorithm (MM-2DFT)

As analyzed in Section 3.2, directly solving Formula (5) incurs a computational complexity of $O(N^2)$, making it impractical for most heterogeneous acceleration hardware, such as GPUs and FPGAs. This approach, however, holds engineering significance for NPUs equipped with dedicated matrix multiplication hardware units.

The direct matrix multiplication algorithm (MM-2DFT) solves Formula (5) using the cube units of the NPU directly. When using NPU cube units for computation, $N \times N$ matrices can be decomposed into multiple 16×16 low-order matrices, and the block matrix multiplication method can be used to obtain the transform result of $N \times N$ matrix results. This is what the NPU excels at, and it can fully use the cube units' computing power. Fortunately, most NPU platforms provide high-order matrix multiplication primitives, such as the general matrix multiplication (gemm) operator provided by the Huawei NPU platform. These NPU platforms are commonly used as coprocessors to execute the gemm operator. During execution, the CPU transfers raw data to the NPU platform, which performs the operator calculation and returns the results to the CPU.

When the cube unit of the NPU performs matrix multiplication, it does not directly calculate complex numbers. Therefore, the real and imaginary parts of the DFT matrix need to be used as inputs for the MM-2DFT algorithm. Then, using the properties of complex multiplication, the complex results can be calculated to obtain the desired output. The specific algorithm for MM-2DFT is shown in Algorithm 1.

To prevent the intermediate results from overflowing during the calculation process of the MM-2DFT algorithm, it is necessary to normalize the input values of X beforehand. Afterward, the Fourier Transform is computed on all the "rows" of X, then the result is transposed, and the Fourier Transform is applied to all the "rows" of the transposed result. Note that the "rows" of the transposed result actually correspond to the "cols" of the original matrix. After completing those two transformations, transposing the result yields the final result of the 2D Fourier Transform. During the calculation process, data copy operations can be parallelized in single-operator calculations like gemm, optimizing the overall algorithm execution time.

4.2. Improved Matrix Divide-and-Conquer-Based FFT Algorithm (MV-2FFT)

The MV-2FFT is an NPU implementation of the FFT based on the matrix divide-and-conquer approach [31]. It first divides the input matrix into several minimum block matrices according to the parity of rows. Then, it calculates the DFT for each minimum matrix block. Finally, it iteratively applies Formula (4) via the bottom-up method to obtain the final result. The pseudo-code for the MV-2FFT is presented in Algorithm 2. The detailed execution process can be described in the following three steps.

First, the input image matrix data are divided into blocks of even matrices and blocks of odd matrices. The specific calculation process of the DFT result can be represented by two steps: (1) calculating the number of iterations as $log_2^N - R$ based on the hyperparameter R and (2) using a recursive algorithm to determine the correspondence between the row numbers of the original input matrix and the smallest block matrices. Based on the results, the corresponding rows are copied into the smallest block matrices.

Algorithm 1: MM-2DFT Algorithm

```
Input: $W_r$ real part of the Twiddle factor matrix;
        $W_i$ imaginary part of the Twiddle factor matrix;
        $X_r$ real part of the matrix to be transformed;
        $X_i$ imaginary part of the matrix to be transformed;
        Inverse whether to perform inverse transformation bool Inverse.
Output: $Y_r$, $Y_i$ resulting matrix
Begin
  normalize $X_r$ and $X_i$, coefficient nN
  for j = 0 to 2 do
    transpose matrix $X_r$, $X_i$ to obtain $X_r^T$, $X_i^T$
    if Inverse == TRUE then
      call gemm complete matrix transpose and conjugate $X_i^T = -1 \times X_i^T$
    end if
    call the gemm operator to do $A = W_r \times X_r^T$, $B = W_i \times X_i^T$,
                                 $C = W_r \times X_i^T$, $D = W_i \times X_r^T$
    if j == 0 then
      call the vector subtraction operator to do $X_r^T = A - B$
      call the vector addition operator to do $X_i^T = C + D$
    end if
    if Inverse == TRUE then
      call the gemm operator and take the conjugate $X_i^T = -1 \times X_i^T$
      call the gemm operator to do $X_r = \frac{1}{N} \times X_r^T$, $X_i = \frac{1}{N} \times X_i^T$
    else if j == 1 then
      call the vector addition and subtraction operator to do $Y_r = A - B$, $Y_i = C + D$
    end if
    if Inverse == TRUE then
      call the vector multiplication operator to do $Y_r = nY_r$
end if
$Y_r = Y_r^T$, $Y_i = Y_i^T$
  end for
    end
```

Then, the DFT results of minimal blocks are calculated separately. The function FFT_1D is called to compute the DFT results for each of the smallest block matrices.

Following this, the final transformation results are gradually computed according to Formula (4) through $log_2^N - R$ iterations. After obtaining the DFT result matrices of the smallest block matrices, the adjacent two result matrices are substituted into Formula (4) to solve for the high-order DFT results. Repeating this process iteratively eventually yields the final Fourier Transform result of the input image.

Finally, the results are copied to the output matrix.

Both the MV-2FFT algorithm and the Cooley-Tukey algorithm share the fundamental concept of matrix divide-and-conquer, but they differ in the partition details (Figure 2). These distinctions include:

- The MV-2FFT algorithm determines the number of iterations and the depth of the matrix partition based on the input hyperparameters. However, the Cooley-Tukey algorithm partitions the matrix to the finest granularity by default, and its iteration time is fixed for log_2^N, where N is the order of the input square matrix. From an algorithm scheduling perspective, MV-2FFT is a more streamlined approach;
- During the iterative computation, the MV-2FFT algorithm operates on matrices, while the Cooley-Tukey algorithm performs butterfly operations on elements of row vectors. The Cooley-Tukey algorithm cannot be implemented on NPUs, as it operates on the element lever, while NPUs can operate only on data blocks. In contrast, the MV-2FFT algorithm can fully utilize the computation power of matrix units.

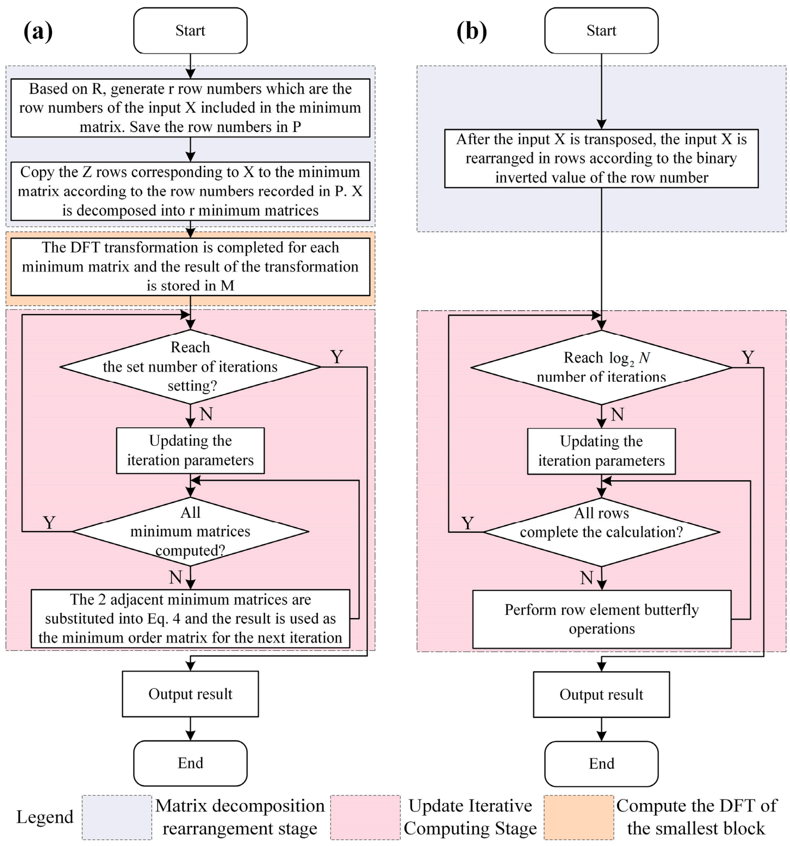
(a)
Start
Based on R, generate r row numbers which are the row numbers of the input X included in the minimum matrix. Save the row numbers in P
Copy the Z rows corresponding to X to the minimum matrix according to the row numbers recorded in P. X is decomposed into r minimum matrices
The DFT transformation is completed for each minimum matrix and the result of the transformation is stored in M
Reach the set number of iterations setting?
Y
N
Updating the iteration parameters
All minimum matrices computed?
Y
N
The 2 adjacent minimum matrices are substituted into Eq. 4 and the result is used as the minimum order matrix for the next iteration
Output result
End
(b)
Start
After the input X is transposed, the input X is rearranged in rows according to the binary inverted value of the row number
Reach log2 N number of iterations
Y
N
Updating the iteration parameters
All rows complete the calculation?
Y
N
Perform row element butterfly operations
Output result
End
Legend
Matrix decomposition rearrangement stage
Update Iterative Computing Stage
Compute the DFT of the smallest block

Figure 2. Flowchart of the MV-2FFT algorithm and Cooley-Tukey algorithm: (**a**) MV-2FFT algorithm and (**b**) Cooley-Tukey algorithm.

```
FUNCTION: FFT_1D
Input: N N × N rotation matrix;
        $X_r$ real part of the input matrix;
        $X_i$ imaginary part of the input matrix;
        $C_t$ nested termination condition.
Output: N × N rotation matrix $Y_r$, $Y_i$
Begin
  read the pre-generated matrices $W_r^N$, $W_i^N$, $D_r^N$, $D_i^N$
  call the gemm operator to calculate $E = W_r^N \times X_r$, $F = W_i^N \times X_i$,
                                      $G = W_r^N \times X_i$, $H = W_r^N \times X_r$
call the vector addition and subtraction operator to do $Y_r = E - F$, $Y_i = G + H$
end
```

The typical approach of calculating $D \times B$ in Algorithm 2 is calling on the gemm operator. However, this calculation can be simplified by employing vector multiplication, considering that D is a diagonal matrix. In detail, the matrix $D = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & \omega_N^1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \omega_N^{\frac{N}{2}-1} \end{bmatrix}_{\frac{N}{2}\times\frac{N}{2}}$ in Formula (3) can be converted to the form $\begin{bmatrix} 1 & 1 & \cdots & 1 \\ \omega_N^1 & \omega_N^1 & \cdots & \omega_N^1 \\ \vdots & \vdots & \ddots & \vdots \\ \omega_N^{\frac{N}{2}-1} & \omega_N^{\frac{N}{2}-1} & \cdots & \omega_N^{\frac{N}{2}-1} \end{bmatrix}_{\frac{N}{2}\times\frac{N}{2}}$, which allows us to replace matrix multiplication with vector multiplication.

Algorithm 2: MV-2FFT Algorithm

```
Input: N N × N rotation matrix;
        $X_r$ real part of the input matrix;
        $X_i$ imaginary part of the input matrix;
        R is a hyperparameter that is set to $\log_2^Z$ when the minimum sub-block matrix is defined as
a Z × Z matrix.
Output: N × N rotation matrix $Y_r$, $Y_i$
Begin
  allocate memory for an array $M\left[2^{(\log_2^N - R)}\right][N \times N]$
  transpose matrices $X_r$, $X_i$ to obtain $X_r^T$, $X_i^T$
  using the recursive method to calculate the assigned row numbers for each minimum sub-block
matrix and fill the results into matrix $P[2^{(\log_2^N - R)}][Z]$
  for i = 0 to $2^{(\log_2^N - R)}$ do
    for j = 0 to Z do
  copy each row of matrices $X_r^T$, $X_i^T$ to matrices $M[i]_r$, $M[i]_i$ based on the values of matrix P
    end for
  end for
  for i = 0 to $2^{(\log_2^N - R)}$ do
    $M[i]_r, M[i]_i = \textbf{FFT\_1D}(Z, M[i]_r, M[i]_i)$
  end for
  for i = $\left(\log_2^N - R\right)$ to 0 do
    for j = 0 to $2^i/2$ do
      call the gemm operator to do $E = D_r^{N/2^i} \times M[2j+1]_r$, $F = D_i^{N/2^i} \times M[2j+1]_i$,
                                   $G = D_r^{N/2^i} \times M[2j+1]_i$, $H = D_r^{N/2^i} \times M[2j+1]_r$
      $DB_r = E - F$
      $DB_i = G + H$
      copy the $N/2^i \times N/2^i$ order matrix $M[2j]_r$ to $A_r$
      copy the $N/2^i \times N/2^i$ order matrix $M[2j]_i$ to $A_i$
      calculate based on Formula (4): $M[j]_r^u = A_r + DB_r$, $M[j]_i^u = A_i + DB_I$,
                                      $M[j]_r^d = A_r - DB_r$, $M[j]_i^d = A_i - DB_i$
    end for
end for
  copy the matrices $\mathbf{M}[\mathbf{0}]_r$, $\mathbf{M}[\mathbf{0}]_i$ matrices to $\mathbf{Y_r}$, $\mathbf{Y_i}$
end
```

We implemented Algorithm 2 by calculating $D \times B$ in Formula (4) via employing a vector multiplication operator rather than a gemm operator. To investigate the effect of this replacement on computation speed and energy efficiency, we implemented another version of the MV-2FFT algorithm with the matrix multiplication operator. In the following sections, we refer to the algorithm using the matrix multiplication operator as MV-2FFTm and the one using the vector multiplication operator as MV-2FFTv.

The flow of the MV-2FFT algorithm is shown in Algorithm 2, where the FFT-1D function is used to calculate the DFT of the smallest block matrix.

5. Results and Discussion

5.1. Experiment Setup

5.1.1. Hardware Platform

We adopted the Atlas 200I DK A2 developer kits made by Huawei in China as the experimental hardware platform to deploy the implementations of our algorithms, which are based on the Ascend 310 series AI processor. The Ascend 310 is one of the most popular NPUs in edge intelligence. Our development and validation of the simplified development kits of the Ascend 310 are conducive to the application and promotion of our algorithm. The AI processor provides four 1 GHz A55 ARM cores and one 500 MHz AI Core. On the software side, the Huawei Atlas 200I DK A2 runs Ubuntu 20.04 and deploys an NPU runtime environment configuration. Three CPUs were defined as control CPUs to run the operating system and applications, while one CPU was dedicated to running AI CPU operators.

5.1.2. Software Flow of the Edge Filter

The combination of high-pass filtering and the Fourier Transform is a common usage. Specifically, the software edge filter uses the high-pass filter to extract contour images. The extraction results can be visualized through the reverse Fourier Transform, which facilitates validating the correctness of algorithms. The software execution stream of the edge filter is shown in Figure 3.

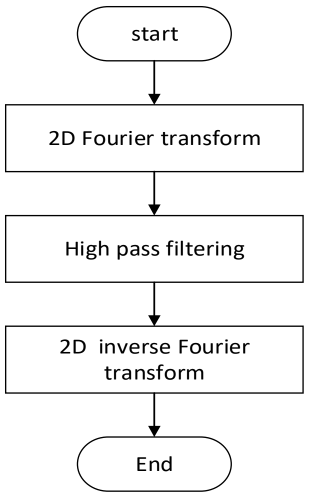

Figure 3. Software flow chart of the edge filter.

We prepared six types of PGM format images with specifications of 64 × 64, 128 × 128, 256 × 256, 512 × 512, 1024 × 1024, and 2048 × 2048 as the input for the Fourier Transform algorithms.

5.1.3. Algorithm and Configuration

To conduct a comprehensive comparison, we benchmarked three implementations of our algorithms running on NPUs against the Cooley-Tukey algorithm running on other platforms for execution speed. One baseline we used was the Cooley-Tukey algorithm based on OpenCL running on the NVIDIA Tegra X2 built-in GPU, referred to as FFT_CL. NVIDIA Tegra X2 is specifically designed for edge applications, and its level of compute power is representative. Another baseline was the Cooley-Tukey algorithm running on an embedded CPU within the NPU, referred to as FFT_CPU.

Note that the FFT_CL was excluded from the energy efficiency evaluation as the different hardware designs interfered with the experiment's results. An additional reason was that we focused more on the impact of the NPU algorithm and internal units on the overall energy efficiency of the board. For the same reason, the FFT_CPU was included in the analysis of CPU energy efficiency as an internal unit of the NPU. That is because CPU units are usually integrated into NPU architectures to support the execution of algorithms that cannot be effectively processed through matrix and vector computations.

For the algorithm configuration, we used the GCC O2 level optimization for the FFT_CPU and set the minimum block matrix size differently for MM-2FFT. These configurations were obtained based on preliminary experimental results to maximize performance. Specifically, the GCC O2 level optimization was used because it accelerates the FFT_CPU algorithm up to three times faster than the non-optimized one, and it outperforms all other optimization levels. Regarding matrices division, we partitioned the input matrix into block matrices whose order was half of the original matrix when the order of input matrices was lower than 2048. Specifically, for an input matrix of order 2048, we set the minimum block matrix size to 512 × 512. We selected those configurations as they are shown to be the optimal ones according to the preliminary experiment. It is worth noting the experiment result shows that fine-grained division of the input matrix may slow down the algorithm execution on the NPU platform. This is because the fine-grained division strategy incurs high scheduling costs. For the configuration items not mentioned above, we used the default settings.

5.2. Acceleration Effect and Discussion

5.2.1. Acceleration Effect of Experiment Results

In this section, we validated the acceleration effect of NPUs on the Fourier Transform following the experimental settings mentioned in the former section. The execution times for performing one edge filter process of various algorithms are shown in Table 1, and their line graphs are shown in Figure 4. The data shown in Table 1 were obtained by executing the edge filter 100 times and calculating the average time consumption. We have bolded the notable outcomes for each set of inputs in Table 1.

Table 1. Time consumption of a single process (ms).

Input Matrix's Order	FFT_CL (Tegra X2)	MM-2DFT	MV-2FFTm	MV-2FFTv	FFT_CPU (with O2)
64	8.41	**1.92**	6.93	6.22	**2.26**
128	11.09	**1.68**	7.35	6.62	10.00
256	22.32	**2.76**	11.60	10.62	44.16
512	37.31	**14.68**	27.89	24.21	197.44
1024	73.99	**67.50**	91.04	82.64	849.90
2048	**271.39**	448.33	483.32	**372.57**	4000.59

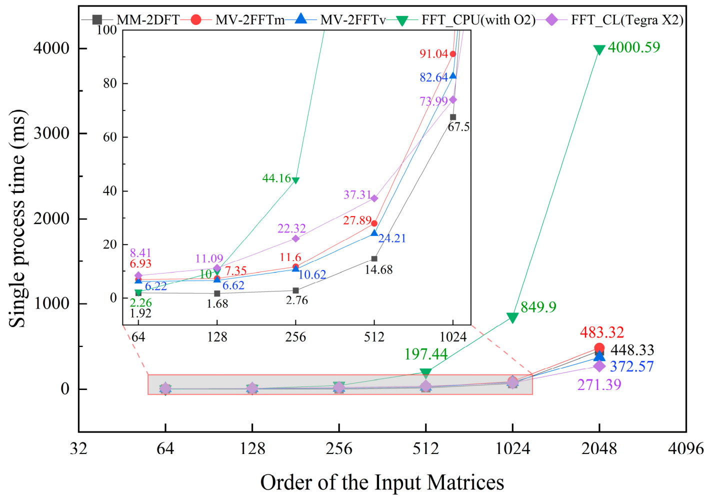

Figure 4. Time consumption of different algorithms with different input scales.

The experiment result showed that the execution time of MM-2DFT, MV-2FFT, FFT-CL, and FFT_CPU increased with the input scale but at different rates. In detail, the result demonstrated that MM-2DFT has the shortest processing time for most input sizes (64 × 64~1024 × 1024), while FFT_CL has the longest processing time with an input size under 256 × 256. As the input size increased, the gap between their runtimes progressively narrowed and eventually reversed. When the input size increased to 2048 × 2048, FFT_CL achieved the highest running speed among the three algorithms, surpassing the second place MV-2FFTv by 27%. Similarly, MV-2FFTv ran 16% faster than MM-2DFT for that input size. Additionally, FFT_CPU showed the second-best performance with the smallest input data size (64 × 64). However, its time consumption increased substantially on larger input data sizes, surpassing other algorithms. This suggested that other computing platforms may have significant advantages over the CPU for this task.

Apart from the growth rate of time consumption, we made another finding. Namely, the MV-2FFTv algorithm consistently exhibited over 10% higher performance compared to MV-2FFTm, and this advantage grew with increasing input sizes.

5.2.2. Analysis of the Acceleration Effect of MM-2DFT

The experiment results showed the MM-2DFT algorithm with the NPU cube unit is more suited for small input sizes than large input sizes. In detail, that algorithm is 4–8 times faster than its competitor, given a DFT input size up to 256 $\times$ 256, and maintains its advantage until the input size exceeds 1024 $\times$ 1024. This attribute to the powerful computing power of the cube unit makes up for the high complexity of MM-2DFT, and the scheduling overhead of MM-2DFT is lower. Consequently, with the small input image size up to 256 $\times$ 256, the MM-2DFT achieves high performance by leveraging the cube unit's powerful computational capabilities of NPU with low scheduling overhead. However, as input size increases, time complexity gradually dominates the execution time. That results in the MM-2DFT's inefficient compared with FFT_CL at input sizes of 2048 $\times$ 2048 or larger. In conclusion, MM-2DFT can achieve high performance in input sizes below 1024 $\times$ 1024 by leveraging the computational power of the cube unit and reducing scheduling overhead. When the input size is not lower than 2048 $\times$ 2048, the fast Fourier algorithm shows performance advantages due to its lower time complexity.

5.2.3. Analysis of the Acceleration Effect of MV-2FFT

We measured the single execution time separately for two implementations of the MV-2FFT algorithm, namely the MV-2FFTm and the MV-2FFTv, and analyzed the results to evaluate the impact of processing unit selection on performance.

1. Analysis of Execution Time Change with Input Size

Compared with MM-2DFT, MM-2FFT exhibited a higher execution time under the order of the input matrix size equal to 1024. Nevertheless, as the size of the input matrix increases, MM-2FFT's execution time increases slower than that of MM-2DFT. Eventually, when the order of the input square matrix reaches 2048, MM-2FFTv's time efficiency surpasses that of MM-2DFT. This observation implies that when the input data size is small, the scheduling overhead involved by the divide-and-conquer method outweighs the potential benefits of that algorithm. As the size of the input matrix increases, however, the calculation complexity advantage of the divide-and-conquer approach used in MM-2FFT gradually becomes more prominent.

2. Analysis of Performance Difference Between MV-2FFTm and MV-2FFTv

When evaluating the two implementations of MV-2FFT, the comparison result demonstrates that MV-2FFTv outperforms MV-2FFTm on all tested input sizes. This performance difference grows as the input size increases, ranging from 10% to 22% improvement as the order of the input square matrix grows from 64 to 2048. This computation time difference is caused by the different handling of those MV-2FFT implementations to calculate $D \times B$ in ALGORITHM 2. Specifically, MV-2FFm calculates multiplication with the cube unit. In contrast, MV-2FFv replaces matrix multiplication with vector multiplication, taking advantage of the sparsity property of the matrix, thereby accelerating computation.

5.3. Energy Consumption Measurement Results and Discussion

5.3.1. Energy Consumption Measurement Results

In this section, we measured the energy consumption of our algorithms and implementations in two usage scenarios.

1. Continuous test results

In the continuous test, we measured energy consumption by continuously processing input images with varying data sizes using edge filters based on MM-2DFT, MV-2FFT, MV-2FFTm, MV-2FFTv, and FFT_CPU. The measurement results of energy consumption of the hardware platform during 1-h operation are shown in Table 2, and their change trend is shown in Figure 5. Like the processing in Table 1, we have bolded the notable outcomes for each set of inputs in Table 2.

Table 2. Entire-board energy consumption during continuous test.

Input Matrix's Order	MM-2DFT		MV-2FFTm		MV-2FFTv		FFT_CPU	
	E	*n*	*E*	*n*	*E*	*n*	*E*	*n*
64	13.1 wh	1,785,504	**13.2 wh**	511,567	12.8 wh	577,538	**12.2 wh**	1,556,041
128	13.2 wh	2,010,560	**13.3 wh**	478,662	12.8 wh	544,520	**12.3 wh**	358,985
256	13.4 wh	1,171,872	**13.4 wh**	301,800	13.0 wh	330,720	**12.4 wh**	81,002
512	13.5 wh	228,840	**13.7 wh**	123,473	13.3 wh	147,448	**12.4 wh**	18,000
1024	13.7 wh	51,864	**14.4 wh**	38,041	14.0 wh	42,144	**12.4 wh**	4206
2048	14.0 wh	7920	**14.8 wh**	7217	14.6 wh	9336	**12.5 wh**	870

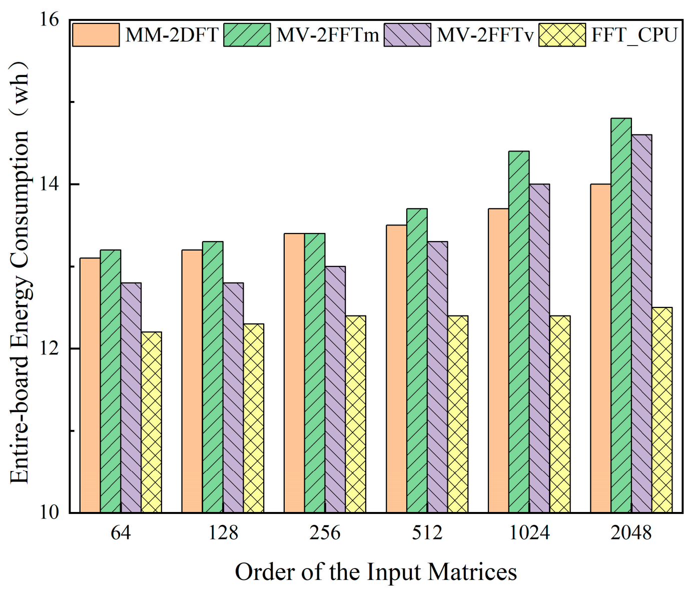

Figure 5. Entire board of energy consumption.

Figure 5 illustrates the trend of the total energy consumption of different algorithms as a function of the input matrix size when executed consecutively. According to the test results, those algorithms can be ranked by total power consumption in the following order from low to high: FFT_CPU, MV-2FFTv, MM-2DFT, and MV-2FFTm. In detail, the FFT_CPU algorithm incurred a mild increase in the board's total power consumption as the input matrix size grew. In contrast, the board's total power consumption of other tested algorithms, especially MV-2FFTm and MV-2FFTv, increased significantly with the input data size.

We defined the calculation method for the energy consumption of a single algorithm computation as Formula (6); the average energy consumption per execution (Ea) is calculated from hardware power consumption (E) and execution count (n) as follows:

$$Ea = E/n \tag{6}$$

Using Formula (6), we were able to calculate the energy consumption of each algorithm for a single execution based on the continuous running test results, as shown in Figure 6.

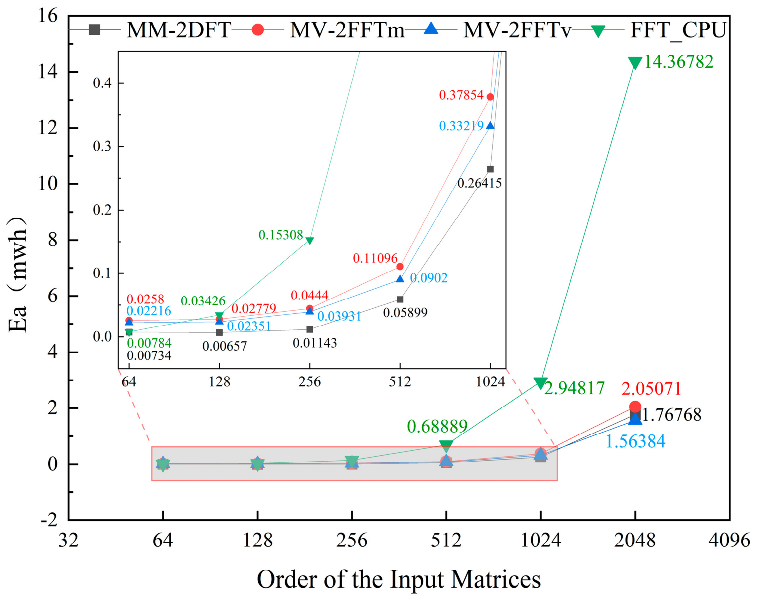

Figure 6. Energy consumption of single execution in continuous test.

Different from Figure 5, Figure 6 shows the single execution energy consumption of those algorithms during the continuous execution test. The trend depicted by the broken lines in the chart indicates a substantial increase in energy consumption per execution of various algorithms as the input matrix size increased, but the growth rates varied significantly. The FFT_CPU algorithm had the highest growth rate in energy consumption with increasing input data size, which significantly exceeded that of other algorithms. Regarding the other algorithms, the MM-2DFT algorithm had the highest energy efficiency for input sizes equal to or lower than 1024 $\times$ 1024, followed closely by MV-2FFv and MV-2FFm algorithms. Once the input size reached 2048 $\times$ 2048, the energy efficiency of the MM-2DFT algorithm was surpassed by that of the MV-2FFTv algorithm.

2. Fixed frame rate test results

For the periodic running test, we invoked the image edge filter using input images of varying sizes. In this test, the processing period was set to 50 ms to simulate an input image stream with a frame rate of 20fps. Due to that process time constraint, we had only used small input data sizes (up to 512 $\times$ 512) that can be processed by most algorithms within one processing period.

The energy consumption of the hardware platform after running for 1-h is displayed in Table 3, and the corresponding bar chart is shown in Figure 7. The bolded data in that table are notable outcomes for each input sets.

Table 3. Entire board of energy consumption during continuous tests.

Input Matrix's Order	MM-2DFT	MV-2FFTm	MV-2FFTv	FFT_CPU
64	**11.8 wh**	12.0 wh	12.0 wh	**11.8 wh**
128	**11.9 wh**	12.1 wh	12.0 wh	**11.9 wh**
256	**11.9 wh**	12.2 wh	12.2 wh	12.4 wh
512	**12.3 wh**	12.9 wh	12.8 wh	---

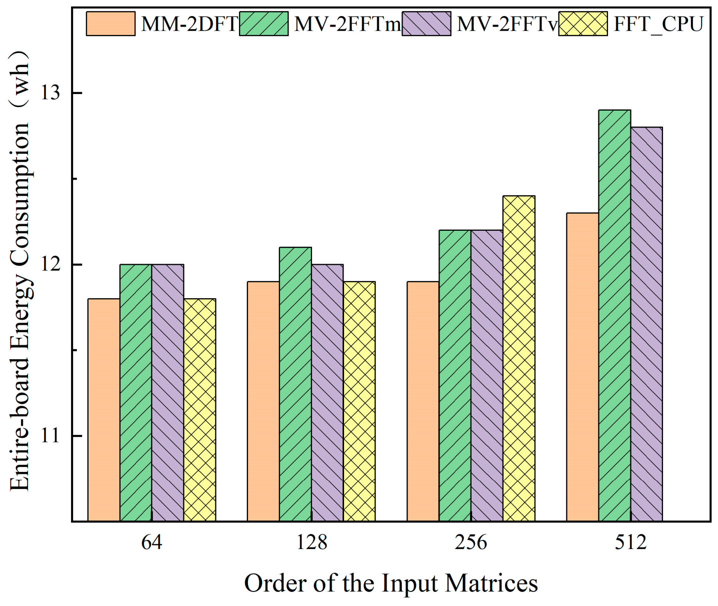

Figure 7. Entire board of energy consumption in the periodic test.

Figure 7 illustrates that the energy consumption of various algorithms generally increased as the order of input matrices increased, yet with different growth rates. Among these algorithms, MM-2DFT was the most energy-efficient overall. FFT_CPU showed low power consumption comparable to MM-2FFT when the input data size was equal to or lower than 128 × 128. However, its power consumption soared to the highest among all the evaluated algorithms when the input data size reached 256 × 256 or higher. Regarding the two implementations of MV-2FFT, MV-2FFTv was more energy-efficient compared to MV-2FFTm but still consumed more time than MM-2DFT on all tested input data sizes.

5.3.2. Discussions of Energy Efficiency for Algorithms

1. Discussions of energy consumption for executing continuously on the entire board

During continuous execution, the power consumption of the FFT_CPU algorithm showed a slight increase as the input data size increased, whereas the NPU algorithm implementations demonstrated a more pronounced rise, as shown in Figure 5. Considering that power consumption has a positive correlation with processor utilization, we monitored the utilization of the CPU and AI Core by those algorithms to investigate the reasons for the above phenomenon. The monitor results showed that, regardless of the input size, FFT_CPU fully occupies a single CPU core but no AI Core. In contrast, the utilization of both the control CPU and AI Core of other algorithms increased as the input size grew, resulting in a significant rise in the power consumption of the whole board. The increase in utilization is attributed to the decrease in the proportion of communication cost between the NPU and CPU to the total overhead as the input size increased. Specifically, algorithms that rely on NPUs require transmitting input data to and results from the NPU, resulting in considerable communication overhead. As the input scale increased, more data were passed to the NPU in a single data pass. That allowed the internal units to participate more fully in the calculation rather than importing and exporting data frequently, which led to a significant increase in NPU utilization.

2. Discussions of energy efficiency for a single execution

The energy efficiency of algorithms is strongly related to their execution time, meaning that optimizing process speed may help to improve energy efficiency. The comparison between Figures 4 and 6 indicates a high degree of consistency between the trends of changes in energy consumption and time consumption of different algorithms concerning input size. Note that lower energy consumption implies higher energy efficiency, while lower time consumption implies higher time efficiency. Hence, that shared trend suggests the speed of a single execution is a critical factor that influences the energy efficiency of the algorithm executing continuously under high load without considering any particular optimization strategy. To conclude, optimizing the execution speed is equivalent to optimizing the energy efficiency to some extent.

3. Analysis of energy efficiency considering static power consumption

Continuous running tests may reflect busy working conditions, but in practical engineering, the frame rate of image input channels is typically fixed. Hence, after completing the computation for one frame, the system enters a waiting task state but still consumes power. Given that under a fixed frame input rate, the total power consumption of the algorithm is the sum of all the execution and standby power consumption, the former is referred to as the dynamic power consumption, while the latter is referred to as the static power consumption. Although the FFT_CPU algorithm has a slightly longer execution time, it does not utilize the extra hardware resources of the AI Core and has a low dynamic power. Thus, the overall power consumption of FFT_CPU is controlled. That is supported by the fact that at those input scales, the power consumption of FFT_CPU is tied with that of MM-2DFT for the lowest. To conclude, despite being less efficient than other NPU algorithms on large inputs, FFT_CPU remains the preferred algorithm for small inputs (up to 128). As the image resolution increases, however, the energy efficiency advantages of NPU algorithms become more noticeable, and they can complete computing tasks faster and with less power consumption compared to CPU algorithms. To summarize, the total power consumption depends not only on the energy efficiency of the algorithms used. Hence, employing the fine-grained device resource energy-saving scheduling algorithms is crucial to leverage the energy efficiency of an algorithm in all scenarios.

5.3.3. Discussions of Energy Efficiency for the Internal Units of the NPU

Next, we analyzed the energy efficiency characteristics of the main acceleration computation units within the NPU, including the cube units, vector units, and embedded CPU. MM-2DFT primarily utilizes the NPU's cube unit. Despite not having any algorithmic advantages over other methods, MM-2DFT shows the best energy efficiency in both energy consumption tests. Hence, the cube unit can be deemed as the most power-efficient computation unit among all the units in NPUs and is suitable for the majority of matrix computation tasks.

The comparison between MV-2FFTm and MV-2FFTv demonstrates that MV-2FFTv is more advantageous in both speed and energy consumption. The main reason for this can be attributed to the optimization technique of replacing a sparse diagonal matrix multiplication with a vector multiplication operation, wherein the vector unit plays a crucial role in enhancing the time and energy efficiency of sparse matrix multiplication operations. Consequently, improving multiplication efficiency in sparse computing scenarios has become a key optimization focus.

Some NPUs usually provide CPU units to support the execution of algorithms that are not compatible with matrix and vector computations. Hence, we consider the CPU to be an internal unit of NPUs to analyze its energy efficiency. Based on the test outcomes of FFT_CPU, it is evident that the CPU possesses prominent advantages in swiftly processing small-scale inputs and keeping low peak power. Therefore, the CPU still emerges as the optimal choice in the above scenarios.

6. Conclusions

This study aims to develop Fourier Transform acceleration algorithms that are well-suited to NPU hardware architecture. This enhancement of the data preprocessing capabilities in NPU-based backpack computers facilitates faster execution of conventional engineering algorithms, supporting various computational service scenarios. Specifically, two NPU algorithms, namely MV-2DFT and MV-2FFT, based on the DFT and the divide-and-conquer approach, were designed. MV-2FFT was implemented in two variants, namely, MV-2FFTm and MV-2FFTv. In terms of the acceleration effect, the MM-2DFT algorithm shows superior performance for small-scale inputs due to the powerful computational ability of cube units. However, its advantage diminishes as the input size increases, compared with FFT_CL and MV-2FFT. As the input size reaches 2048 $\times$ 2048, FFT_CL and MV_2FFT outperform MM-2DFT, primarily attributed to their lower computational complexity. Regarding energy efficiency, it is illustrated that the cube units have the highest energy efficiency among the NPU internal units. However, utilizing CPU or vector units may be advantageous in optimizing speed and reducing energy consumption in specific scenarios, such as handling small-scale inputs and performing sparse matrix computations.

In future work, there are plans to explore methods for optimal energy-efficient scheduling strategies that automatically select different processors and algorithms and determine the hyperparameters of MV-2FFT for Fourier Transform processing on NPU platforms. Meanwhile, NPU acceleration methods for some other preprocessing algorithms, like the discrete wavelet transform, will also be investigated. Addressing the identified issue in this paper, the efficiency of NPUs for sparse matrix multiplication will be optimized by proposing a more general sparse matrix multiplication optimization method. This aims to enhance the energy efficiency of NPUs in sparse scenarios based on the energy-efficient underlying arithmetic.

Author Contributions: Conceptualization, D.W.; methodology, Q.L., D.Z. and Y.F.; software, Q.L.; formal analysis, D.Z.; investigation, Y.F.; resources, D.W.; data curation, Y.F.; writing—original draft preparation, Q.L. and D.Z.; writing—review and editing, Y.F.; supervision, D.W.; project administration, D.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China (No. 62171155).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Acknowledgments: We would like to express our gratitude to the members and staff of the Fault Tolerance and Mobile Computing Research Center at Harbin Institute of Technology.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Sipola, T.; Alatalo, J.; Kokkonen, T.; Rantonen, M. Artificial Intelligence in the IoT Era: A Review of Edge AI Hardware and Software. In Proceedings of the 2022 31st Conference of Open Innovations Association (FRUCT), Helsinki, Finland, 27–29 April 2022; IEEE: New York, NY, USA; pp. 320–331.
2. Su, W.; Li, L.; Liu, F.; He, M.; Liang, X. AI on the Edge: A Comprehensive Review. *Artif. Intell. Rev.* **2022**, *55*, 6125–6183. [CrossRef]
3. Tan, T.; Cao, G. FastVA: Deep Learning Video Analytics through Edge Processing and NPU in Mobile. In Proceedings of the IEEE INFOCOM 2020—IEEE Conference on Computer Communications, Toronto, ON, Canada, 6–9 July 2020; IEEE: New York, NY, USA, 2020; pp. 1947–1956.
4. Tan, T.; Cao, G. Efficient Execution of Deep Neural Networks on Mobile Devices with NPU. In Proceedings of the 20th International Conference on Information Processing in Sensor Networks (co-located with CPS-IoT Week 2021), Nashville, TN, USA, 18–21 May 2021; ACM: New York, NY, USA, 2021; pp. 283–298.
5. Jouppi, N.P.; Young, C.; Patil, N.; Patterson, D.; Agrawal, G.; Bajwa, R.; Bates, S.; Bhatia, S.; Boden, N.; Borchers, A.; et al. In-Datacenter Performance Analysis of a Tensor Processing Unit. In Proceedings of the 44th Annual International Symposium on Computer Architecture, Toronto, ON, Canada, 24–28 June 2017; ACM: New York, NY, USA, 2017; pp. 1–12.
6. Xue, Y.; Liu, Y.; Nai, L.; Huang, J. V10: Hardware-Assisted NPU Multi-Tenancy for Improved Resource Utilization and Fairness. In Proceedings of the 50th Annual International Symposium on Computer Architecture, Orlando, FL, USA, 17–21 June 2023; ACM: New York, NY, USA, 2023; pp. 1–15.
7. Su, F.; Liu, C.; Stratigopoulos, H.-G. Testability and Dependability of AI Hardware: Survey, Trends, Challenges, and Perspectives. *IEEE Des. Test* **2023**, *40*, 8–58. [CrossRef]
8. Wang, Y.E.; Wei, G.-Y.; Brooks, D. Benchmarking TPU, GPU, and CPU Platforms for Deep Learning. *arXiv* **2019**, arXiv:1907.10701.

9. Chen, T.; Du, Z.; Sun, N. DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. *ACM SIGARCH Comput. Arch. News* **2014**, *42*, 269–284. [CrossRef]
10. Chen, Y.; Luo, T.; Liu, S.; Zhang, S.; He, L.; Wang, J.; Li, L.; Chen, T.; Xu, Z.; Sun, N.; et al. DaDianNao: A Machine-Learning Supercomputer. In Proceedings of the 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, UK, 13–17 December 2014; IEEE: New York, NY, USA, 2014; pp. 609–622.
11. Fang, W.-C.; Wang, K.-Y.; Fahier, N.; Ho, Y.-L.; Huang, Y.-D. Development and Validation of an EEG-Based Real-Time Emotion Recognition System Using Edge AI Computing Platform With Convolutional Neural Network System-on-Chip Design. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2019**, *9*, 645–657. [CrossRef]
12. Wang, S.-T.; Li, I.-H.; Wang, W.-Y. Human Action Recognition of Autonomous Mobile Robot Using Edge-AI. *IEEE Sens. J.* **2023**, *23*, 1671–1682. [CrossRef]
13. Wang, K.; Chen, C.-M.; Hossain, M.S.; Muhammad, G.; Kumar, S.; Kumari, S. Transfer Reinforcement Learning-Based Road Object Detection in next Generation IoT Domain. *Comput. Netw.* **2021**, *193*, 108078. [CrossRef]
14. Hashir, M.; Khalid, N.; Mahmood, N.; Rehman, M.A.; Asad, M.; Mehmood, M.Q.; Zubair, M.; Massoud, Y. A TinyML Based Portable, Low-Cost Microwave Head Imaging System for Brain Stroke Detection. In Proceedings of the 2023 IEEE International Symposium on Circuits and Systems (ISCAS), Monterey, CA, USA, 21–25 May 2023; IEEE: New York, NY, USA, 2023; pp. 1–4.
15. Hu, Y.; Lu, L.; Li, C. Memory-Accelerated Parallel Method for Multidimensional Fast Fourier Implementation on GPU. *J. Supercomput.* **2022**, *78*, 18189–18208. [CrossRef]
16. Balabanova, I.S.; Georgiev, G.I. Image Recognition by FFT, Artificial Intelligence and k-Nearest Neighbors Approach. *J. Phys. Conf. Ser.* **2022**, *2339*, 012008. [CrossRef]
17. Sedik, A.; Marey, M.; Mostafa, H. WFT-Fati-Dec: Enhanced Fatigue Detection AI System Based on Wavelet Denoising and Fourier Transform. *Appl. Sci.* **2023**, *13*, 2785. [CrossRef]
18. Seyed Mahmoud, S.M.A.; Faraji, G.; Baghani, M.; Hashemi, M.S.; Sheidaei, A.; Baniassadi, M. Design of Refractory Alloys for Desired Thermal Conductivity via AI-Assisted In-Silico Microstructure Realization. *Materials* **2023**, *16*, 1088. [CrossRef] [PubMed]
19. Hashemi, M.S.; Safdari, M.; Sheidaei, A. A Supervised Machine Learning Approach for Accelerating the Design of Particulate Composites: Application to Thermal Conductivity. *Comput. Mater. Sci.* **2021**, *197*, 110664. [CrossRef]
20. Le Ba, N.; Kim, T.T.-H. An Area Efficient 1024-Point Low Power Radix-2 2 FFT Processor With Feed-Forward Multiple Delay Commutators. *IEEE Trans. Circuits Syst. Regul. Pap.* **2018**, *65*, 3291–3299. [CrossRef]
21. Zhao, Y.; Liu, F.; Ma, W.; Li, H.; Peng, Y.; Wang, C. MFFT: A GPU Accelerated Highly Efficient Mixed-Precision Large-Scale FFT Framework. *ACM Trans. Archit. Code Optim.* **2023**, *20*, 1–23. [CrossRef]
22. Ayala, A.; Tomov, S.; Stoyanov, M.; Haidar, A.; Dongarra, J. Performance Analysis of Parallel FFT on Large Multi-GPU Systems. In Proceedings of the 2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Lyon, France, 30 May–3 June 2022; IEEE: New York, NY, USA, 2022; pp. 372–381.
23. Lee, J.; Kang, D.; Ha, S. S3NAS: Fast NPU-Aware Neural Architecture Search Methodology. *arXiv* **2020**, arXiv:2009.02009.
24. Lee, J.; Park, J.; Lee, S.; Kung, J. Implication of Optimizing NPU Dataflows on Neural Architecture Search for Mobile Devices. *ACM Trans. Des. Autom. Electron. Syst.* **2022**, *27*, 1–24. [CrossRef]
25. Rapp, M.; Krohmer, N.; Khdr, H.; Henkel, J. NPU-Accelerated Imitation Learning for Thermal- and QoS-Aware Optimization of Heterogeneous Multi-Cores. In Proceedings of the 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), Antwerp, Belgium, 14–23 March 2022; IEEE: New York, NY, USA, 2022; pp. 584–587.
26. Kim, H.; Park, H.; Kim, T.; Cho, K.; Lee, E.; Ryu, S.; Lee, H.-J.; Choi, K.; Lee, J. GradPIM: A Practical Processing-in-DRAM Architecture for Gradient Descent. In Proceedings of the 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), Seoul, Republic of Korea, 27 February–3 March 2021; IEEE: New York, NY, USA, 2021.
27. Lin, X.; Liu, R.; Xie, J.; Wei, Q.; Zhou, Z.; Chen, X.; Huang, Z.; Lu, G. Online Scheduling of CPU-NPU Co-Inference for Edge AI Tasks. In Proceedings of the 2023 IEEE Wireless Communications and Networking Conference (WCNC), Glasgow, UK, 26–29 March 2023; IEEE: New York, NY, USA, 2023; pp. 1–6.
28. Kang, D.; Oh, J.; Choi, J.; Yi, Y.; Ha, S. Scheduling of Deep Learning Applications Onto Heterogeneous Processors in an Embedded Device. *IEEE Access* **2020**, *8*, 43980–43991. [CrossRef]
29. Yao, L. *Ascend AI Processor Architecture and Programming: Principles and Applications of CANN*; Tsinhua University Press: Beijing, China, 2019.
30. Tsuchiyama, R.; Nakamura, T.; Lizuka, T.; Asahara, A.; Miki, S. *The OpenCL Programming Book*; Fixstars Corporation: Tokyo, Japan, 2009.
31. Gorlatch, S.; Bischof, H. Formal Derivation of Divide-and-Conquer Programs: A Case Study in the Multidimensional FFT's. In *Formal Methods for Parallel Programming: Theory and Applications. Workshop at IPPS*; University of Passau: Passau, Germany, 1997; pp. 80–94.