

# GPU-Based Parallel Computing Methods for Medical Photoacoustic Image Reconstruction

Xinyao Yi

xyi2@uncc.edu

University of North Carolina at Charlotte  
Charlotte, North Carolina, USA

Yuxin Qiao

yq83@nau.edu

Northern Arizona University  
Flagstaff, Arizona, USA

## ABSTRACT

Recent years have witnessed a rapid advancement in GPU technology, establishing it as a formidable high-performance parallel computing technology with superior floating-point computational capabilities compared to traditional CPUs. This paper explores the application of this technology in the field of photoacoustic imaging, an emerging non-destructive testing technique in biomedical engineering characterized by its high contrast, resolution, and penetration depth. We conduct a data parallelism analysis targeting the computationally intensive image reconstruction segment of photoacoustic imaging. By parallelizing the serial code for iterative reconstruction and optimizing memory access, we achieve significant improvements in processing speed. Our experiments compare the imaging speeds of vascular images reconstructed using CPUs and GPUs, with the results visualized using Matlab. The findings demonstrate that, while maintaining data accuracy, GPU parallel computing methods can markedly accelerate photoacoustic image reconstruction. This acceleration has the potential to facilitate the broader adoption of photoacoustic imaging in applications such as hemodynamic monitoring, clinical disease diagnosis, and drug development.

## KEYWORDS

parallel computing, CUDA, photoacoustic image, reconstruction

## 1 INTRODUCTION

As science and technology advance, we encounter increasingly complex graphics computing problems and significant data machine losses [3]. Concurrently, the computational demands in scientific fields have escalated, calling for enhanced high-performance computing solutions [2, 5]. Graphics Processing Units (GPUs), designed specifically for image computations, comprise numerous computing units, offering a rapid parallel computing framework for large-scale data processing with superior floating-point capabilities.

,,

.

In recent years, GPU parallel computing technology has seen rapid development and widespread application across various domains, primarily due to its cost-effectiveness and efficiency in handling large-scale computations [17]. One such application is in the field of photoacoustic imaging, an emerging non-destructive testing technique in biomedical engineering. Photoacoustic imaging is distinguished by its high contrast, resolution, and penetration depth, making it a promising medical imaging modality with significant potential for early diagnosis of cancer and cardiovascular and cerebrovascular diseases [14].

However, the vast data volume and complexity of photoacoustic image processing pose challenges that exceed the capabilities of traditional CPUs [9], thereby impeding the progress of medical photoacoustic imaging. Addressing this issue requires efficient medical image processing and analysis techniques to accurately and swiftly analyze massive image datasets. The rapid evolution of GPU parallel computing offers a promising solution to this challenge by leveraging the immense computational power of GPUs for handling the characteristics of massive data, complex algorithms, and high parallelism inherent in medical images.

This paper delves into GPU parallel computing methods, focusing on the parallel acceleration of iterative reconstruction algorithms during the image reconstruction process. By significantly enhancing the reconstruction speed, this study lays the groundwork for the future realization of real-time iterative reconstruction of photoacoustic images, thereby advancing the field of medical imaging.

## 2 PARALLEL COMPUTING ON GPU

### 2.1 Parallel Computing on GPU

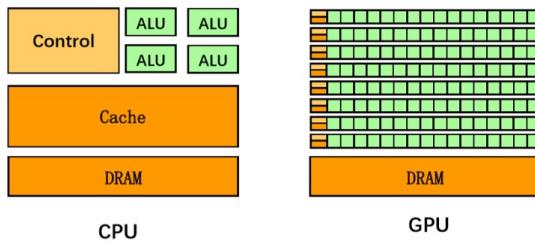
*2.1.1 Emergence and Development of GPU Technology.* The Graphics Processing Unit (GPU) stands as a pivotal element of a graphics card, predominantly dictating its performance tier and capabilities. In 1999, NVIDIA Corporation released the GeForce 256 graphics processor, which established the conceptual foundation of the modern GPU. This pivotal innovation ignited a demand for increasingly sophisticated applications, propelling a sustained and dynamic expansion of the industry. Through iterative advancements, the programmability of GPUs has been progressively augmented,

leading to their utilization across a diversifying array of computational tasks.

In the GPU's nascent phase before 1999, programmability was notably absent, confining their role to hardware-accelerated image rendering [11]. The advent of the second-generation GPU, spanning from 1999 to 2002, witnessed a decentralization of functionalities from the CPU, thereby broadening the application spectrum of GPUs. The year 2001 marked a significant milestone with NVIDIA's GeForce3 and ATI's Radeon 8500, which redefined graphics pipelines as stream processors, endowing them with vertex-level programmability and, albeit to a lesser extent, pixel-level programmability [10].

The post-2002 era heralded the third-generation GPUs characterized by a substantial reduction in the complexity associated with general-purpose computational programming. A paradigm shift occurred in 2007 with NVIDIA's introduction of the CUDA (Compute Unified Device Architecture) programming environment. This development metamorphosed the GPU into a versatile, programmable, high-performance parallel computing platform, inaugurating a new era dedicated to general-purpose computational applications.

**2.1.2 Computational Advantages of GPUs.** The computational superiority of GPUs over CPUs stems from the inherent differences in their design philosophies and operational objectives. Figure 1 illustrates the architectures of a CPU and a GPU. In a CPU, a significant portion of the chip (approximately 70%) is dedicated to building multi-level caches and control units related to instruction-level parallelism, including out-of-order execution, branch prediction, and other logic-related components. Conversely, the arithmetic units occupy a relatively smaller fraction of the chip. On the other hand, a GPU is essentially a massive array of computing units, with a greater emphasis on arithmetic units, including integer and floating-point multiply-add units, and specialized computing units.



**Figure 1: CPU and GPU Structures**

Moreover, unlike CPUs, GPUs possess higher-bandwidth memory, commonly referred to as video memory (VRAM), which enables them to perform exceptionally well in applications requiring high throughput. Consequently, while CPUs

excel at handling instructions with complex logical interdependencies, GPUs have a distinct advantage in processing data that can be executed in parallel without dependencies.

### 2.1.3 GPU+CPU Heterogeneous Parallel Computing Model.

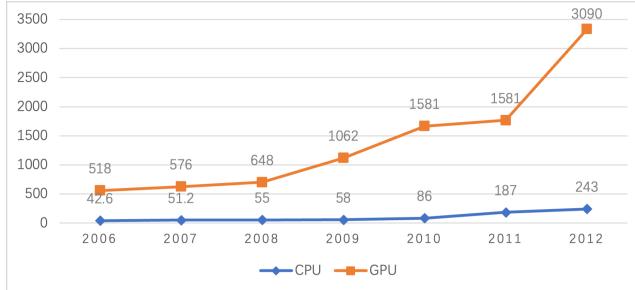
The heterogeneous computing model, which synergistically combines GPUs with CPUs, capitalizes on their complementary processing strengths. Initially designed for high-throughput graphics rendering, GPUs possess inherent parallel processing capabilities that have significantly evolved beyond their original scope. The convergence of enhanced programmability and computational prowess has extended GPU applications to a broad spectrum of computational tasks.

In this integrated model, the serial and complex logical processing tasks are adeptly handled by the CPU, leveraging its optimized cores for sequential execution. Conversely, the GPU is employed for its extensive array of compute units, adept at parallel processing large-scale data. Such a division of labor enables each processor to function in its area of competence, thereby maximizing overall system efficiency. This collaborative approach can yield computational performance gains by orders of magnitude over CPU-only systems.

In 2007, NVIDIA Corporation developed the Compute Unified Device Architecture (CUDA), a paradigm-shifting framework that catalyzed the widespread adoption and growth of GPU parallel computing [4]. As an extension of the C language, CUDA enables programming of GPU code using standard C, rendering the code compatible with both CPUs and GPUs. This breakthrough eliminated the need for developers to master complex shading languages or graphical primitives for GPU programming. Consequently, a model for heterogeneous hybrid parallel computing combining GPU and CPU was proposed.

The combination of CPU and GPU harnesses the strengths of both processors: the CPU, with its cores optimized for sequential processing, handles the serial components of a program, while the GPU, comprising thousands of compute units, manages the parallel tasks. This collaborative computational model significantly enhances the capabilities of both CPUs and GPUs, achieving efficiency gains that can surpass traditional CPU-only computations by several orders of magnitude.

As depicted in Figure 2, the gap in computational capabilities between CPUs and GPUs has widened with technological advancements. By 2012, the speed difference had already reached approximately 12-fold. This disparity underscores the rapid pace at which GPU technology has evolved, significantly outpacing the improvements in CPU performance. The increasing parallel processing power of GPUs makes them an ideal choice for tasks requiring high computational throughput, further solidifying their advantage in various applications, including medical photoacoustic imaging.



**Figure 2: Peak performance of CPU and GPU (measured in billions of floating-point operations per second, or gigaflops)**

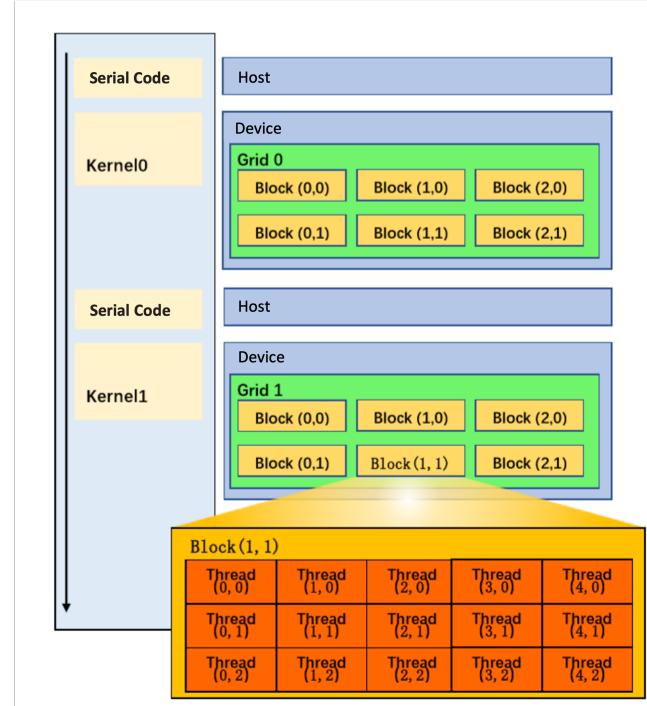
## 2.2 CUDA

Since its introduction, CUDA (Compute Unified Device Architecture) has experienced rapid development, becoming a cornerstone in leveraging the power of GPUs. CUDA allows developers to efficiently utilize GPUs using mainstream programming languages such as C, C++, and Fortran, without the need to learn new languages. This accessibility has significantly lowered the barrier to entry for GPU programming, enabling a wider range of researchers and developers to harness the parallel computing capabilities of GPUs for various applications, including medical image processing and scientific computations. CUDA's comprehensive ecosystem, which includes libraries, development tools, and support forums, further facilitates the development and optimization of GPU-accelerated applications.

**2.2.1 Programming Model.** Within the CUDA architecture, the CPU is designated as the host and the GPU as the device, with both working in conjunction to perform computations. CUDA code is bifurcated into two segments: standard C code executed on the CPU and parallel code, known as a kernel, executed on the GPU and compiled using NVIDIA's nvcc compiler.

Figure 3 illustrates the operational principle of the GPU under the CUDA framework. On the GPU, invoking a kernel function initiates a grid, with each kernel corresponding to a grid. Each grid is composed of one or more thread blocks (blocks), executed on a Streaming Multiprocessor (SM); within each thread block, there are multiple threads (threads) that run on the SM's Compute Units (SPs).

Every thread block is assigned a two-dimensional identifier via CUDA's specific keywords `blockIdx.x` and `blockIdx.y`. All thread blocks are organized in the same manner and contain an equal number of threads. There are two levels of parallelism during execution: blocks within the grid operate in parallel, as do threads within each block. While thread blocks cannot communicate with one another, threads within

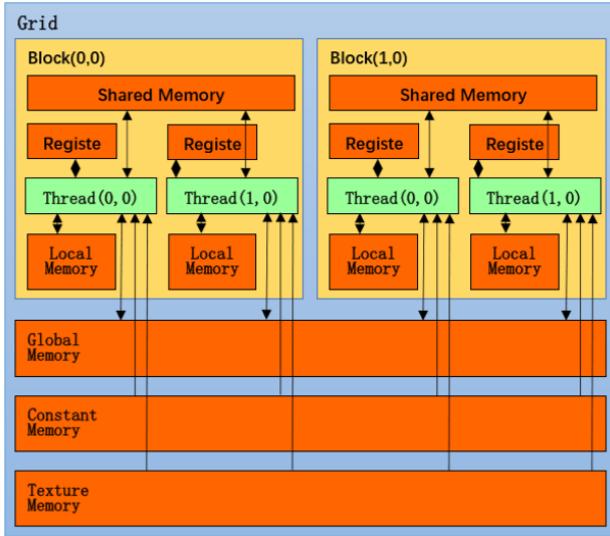


**Figure 3: CUDA Programming Model**

a single block can communicate and synchronize via shared memory (shared memory).

**2.2.2 Memory Model.** In CUDA, distinct memory spaces exist for the host and the device. Consequently, when executing a kernel on the device, programmers must transfer data from host memory to the allocated device memory space. Subsequently, upon completion of kernel execution on the device, results must be transferred back to the host, and the device's memory space must be released.

Figure 4 illustrates the memory model of a GPU, encompassing various memory types. Each thread possesses private registers and local memory. Shared memory is present within each thread block, facilitating communication among threads within the same block. Global memory resides off-chip on the GPU board and is accessible for reading and writing by all threads in a grid. Constant memory and texture memory are restricted to reading operations by threads. Register access incurs nearly zero latency, and accessing shared memory without bank conflicts has latency comparable to register access. In contrast, global memory in the GPU experiences higher latency, approximately 400-800 clock cycles. Consequently, when designing a program, it is advisable to utilize shared memory for data storage to reduce data retrieval time and enhance overall program execution speed. Notably, texture memory is well-suited for image processing operations



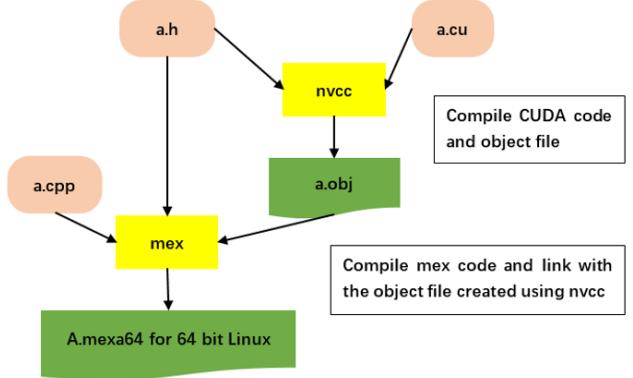
**Figure 4: GPU Hydrogenous Memory Architecture**

due to its ability to accelerate access to large amounts of data with random and non-aligned access patterns.

Memory access latency constitutes a significant limiting factor for GPU processing speed [15]. Effective memory planning is crucial when designing a program to achieve maximum access bandwidth. Various methods, such as switching between different blocks on a Streaming Multiprocessor (SM), storing multiple block contexts in an SM, and scheduling another block for execution on the core when one block is suspended due to latency, can be employed to hide memory access latency [16].

**2.2.3 Integration of CUDA with MATLAB for Enhanced Parallel Computing.** MATLAB demonstrates excellent performance in matrix operations, plotting functions and data, and interfacing with other programming languages. It is primarily used in fields such as engineering computation, control design, signal processing, and communications. With the increasing scale of data computation, many laboratories and research institutions are turning to parallel computing to improve computational speed. MATLAB provides a Parallel Computing Toolbox that can leverage GPUs to enhance code execution speed. However, the Parallel Computing Toolbox has various constraints and limitations, preventing GPUs from fully leveraging their high-performance advantages and resources from being utilized to their full potential. Therefore, combining CUDA with MATLAB programming provides an excellent approach to problem-solving. Assignments, conditional structures, and small-scale matrix operations can

be handled by MATLAB, while CUDA can be used for large-scale, time-consuming matrix operations. During this process, MATLAB's Parallel Computing Toolbox can also assist in performing some small-scale matrix operations, resulting in the best acceleration effect for the program.



**Figure 5: Compilation process of kernel function with MATLAB**

In MATLAB, the CUDA compilation process is illustrated in Figure 5. Firstly, three files need to be compiled in the CUDA model, namely a.h, a.cu, and a.cpp files (where "a" represents the function to be compiled). Using the CUDA built-in compiler nvcc, the a.h and a.cu files are compiled into an a.obj file. After compiling the a.obj file, the mex compiler in MATLAB is used to compile the a.cpp and a.obj files into a callable function A. Once the compilation is complete, function A can be directly called in MATLAB.

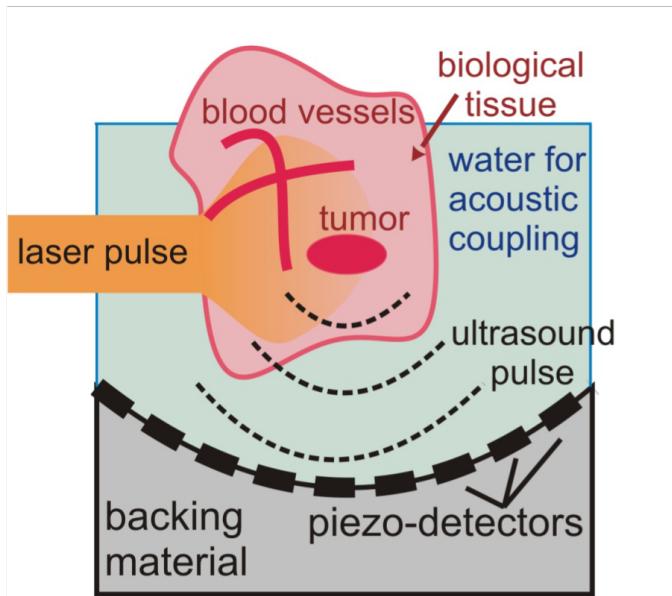
### 3 PHOTOAUDIOUSTIC IMAGING

#### 3.1 Introduction to Medical Photoacoustic Imaging

The domain of biomedical imaging has experienced remarkable advancement since the late 19th century. During this period, various imaging modalities emerged, yet many were either exorbitantly costly or posed significant risks to human health [6]. Over the last two decades, the advent of photoacoustic imaging has addressed these challenges. This modality synergizes the high contrast of optical imaging with the deep penetration capabilities of ultrasound imaging, offering a cost-effective, high-quality, and non-invasive imaging solution that has received widespread recognition [12].

The foundational concept of photoacoustic imaging traces back to the discovery of the photoacoustic effect by Alexander Graham Bell in 1880 [1]. The subsequent progress in laser technology during the 1960s propelled the application of photoacoustic techniques in industrial and scientific domains. By the late 20th century, this technology was harnessed for

biomedical imaging, culminating in the generation of the first photoacoustic medical images. Since then, the evolution of photoacoustic imaging technology has been swift. In comparison to conventional ultrasound imaging, it offers superior optical contrast and provides critical physiological parameters such as hemoglobin oxygen saturation. Moreover, it achieves enhanced spatial resolution for deep tissue imaging relative to pure optical imaging modalities. Despite its relatively recent inception, photoacoustic imaging has been extensively applied in various preclinical imaging scenarios, attracting significant attention [7, 18]. This technique typically utilizes unfocused ultrasound transducers with mechanical scanning or ultrasound transducer arrays for image data acquisition [8, 13], followed by the application of reconstruction algorithms to retrieve the optical properties of the imaged object.



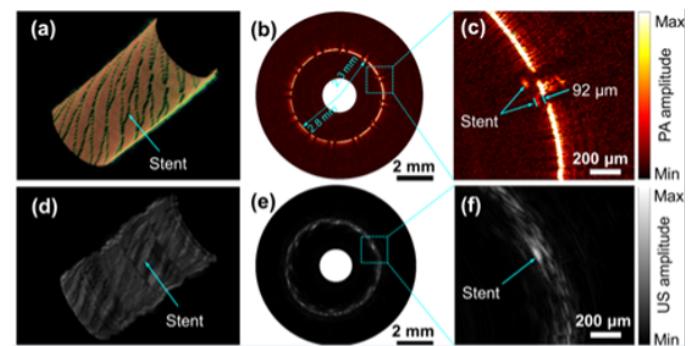
**Figure 6: Principle diagram of the photoacoustic effect.**

The core principle of biomedical photoacoustic imaging is illustrated in Figure 6. A wide-beam, short-pulse laser irradiates biological tissue, leading to the absorption of light energy and subsequent thermal expansion, which generates pressure waves and produces the photoacoustic signal. Ultrasound detectors placed on the surface of the tissue capture these emitted ultrasound waves. By analyzing the detected photoacoustic signals, the distribution of light energy absorption within the tissue is reconstructed to create an image. The differential light energy absorption properties of various biological tissues enable photoacoustic imaging to produce high-contrast structural images. Additionally, when integrated with photoacoustic spectroscopy, this technique

facilitates the quantitative assessment of changes in tissue components and offers a comprehensive depiction of minute tissue lesions and crucial physiological parameters, thereby enhancing functional imaging. These unique advantages render photoacoustic imaging particularly valuable for the early diagnosis of conditions such as breast cancer and prostate cancer.

### 3.2 Advancements and Applications of Medical Photoacoustic Imaging

Medical photoacoustic imaging has become an indispensable part of modern medicine, with numerous scholars at home and abroad conducting in-depth research on this technology [8, 13]. In March 2014, the research group led by Liang Song at the Institute of Biomedical and Health Engineering, Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, developed an intravascular optical-resolution photoacoustic tomography (OR-PAT) system with an imaging resolution of up to 19.6 micrometers. This technology can provide information reflecting important physiological functions such as tissue composition and inflammation activity, playing a significant role in guiding the interventional diagnosis and treatment of cardiovascular diseases.

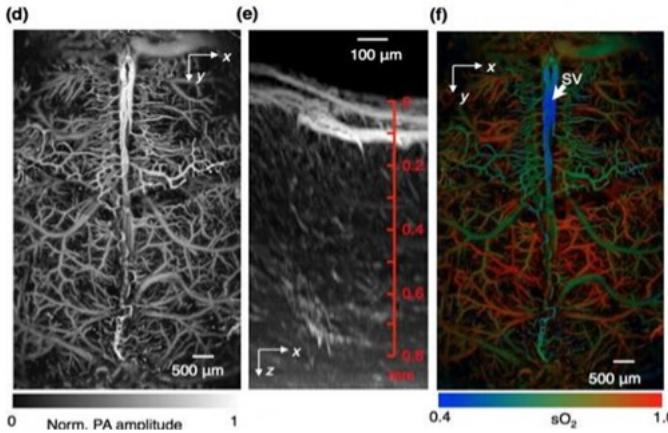


**Figure 7: Evaluation of stent apposition using high-resolution intravascular photoacoustic microscopy (a-c) and intravascular ultrasound imaging (d-f).**

In March 2015, Dr. Lihong Wang and his research team at Washington University in St. Louis developed a fast-functional photoacoustic microscopy (PAM) technique that can detect blood flow, blood oxygen, oxygen metabolism, and other functions in the living mouse brain.

### 3.3 Principle of Photoacoustic Computed Tomography (PACT)

Photoacoustic computed tomography (PACT) allows the acquisition of photoacoustic images. According to the theory



**Figure 8: Images of mouse brain captured by fast-functional photoacoustic microscopy technology:** (d) is the vascular system within the entire skull projected onto the two-dimensional x-y plane; (e) is an enhanced imaging image of a typical brain vascular system projected onto the two-dimensional x-z plane; (f) is an image of mouse brain hemoglobin oxygen saturation captured by photoacoustic microscopy, taken using a new method based on single wavelength and pulse width with two laser beams.

of photoacoustic generation [?], the acoustic pressure  $p(\mathbf{r}, t)$  at position  $\mathbf{r}$  and time  $t$  satisfies the wave equation:

$$\nabla^2 p(\mathbf{r}, t) - \frac{1}{c^2} \frac{\partial^2 p(\mathbf{r}, t)}{\partial t^2} = \frac{\partial^2 p_0(\mathbf{r})}{\partial t^2}, \quad (1)$$

where  $p_0(\mathbf{r})$  is the initial photoacoustic pressure induced by the laser or electromagnetic pulse, and  $c$  is the speed of light. By solving the wave equation, the forward model using prediction can be written as:

$$p(\mathbf{r}, t) = \text{FWD}[p_0(\mathbf{r})], \quad (2)$$

where FWD represents the forward operator, and  $p_0(\mathbf{r})$  is the initial pressure distribution.

The time-dependent wave equation can be transformed into the time-frequency domain by taking the Fourier transform of  $p(\mathbf{r}, t)$  to obtain  $P(\mathbf{r}, \omega)$ , where  $\omega$  is the angular frequency:

$$\nabla^2 P(\mathbf{r}, \omega) + \frac{\omega^2}{c^2} P(\mathbf{r}, \omega) = -\omega^2 P_0(\mathbf{r}), \quad (3)$$

where  $P_0(\mathbf{r})$  is the Fourier transform of the initial pressure  $p_0(\mathbf{r})$ , and  $\omega$  is equal to the angular frequency corresponding to the signal's time frequency. Based on equations (2) and (3), the time-frequency domain forward model can be expressed as:

$$P(\mathbf{r}, \omega) = \text{FWD}_\omega[P_0(\mathbf{r})], \quad (4)$$

where  $\text{FWD}_\omega$  represents the forward operator in the frequency domain.

### 3.4 The Principle of Iterative Reconstruction for Photoacoustic Imaging

Photoacoustic image reconstruction is a core part of photoacoustic imaging technology. The quality of the reconstruction method directly affects the quality of the photoacoustic image and the speed of image acquisition. The most common reconstruction method currently is the Back-Projection (BP) algorithm, which has high resolution and relatively fast imaging speed. However, it has certain limitations in practical applications. When the projection data is unevenly distributed, sparse, or when there is significant noise, the BP method's reconstruction effect is poor, leading to significant artifacts in the results. Thus, Iterative Reconstruction (IR) methods have emerged. Iterative reconstruction algorithms are suitable for reconstructing high-quality images from fewer projection data. However, algebraic iterative reconstruction algorithms are slow because they require multiple iterations to converge and require large data storage, making their lengthy processing time the biggest obstacle to their application. Therefore, improving the speed of iterative reconstruction has important practical value.

**3.4.1 Compressed Sensing.** Compressed Sensing (CS) is a method for recovering sparse signals from undersampled measurements, widely used in CT and Diffuse Optical Tomography. Numerous studies have certified that reconstruction techniques based on CS can recover photoacoustic signals at sampling rates lower than what is demanded by the Nyquist sampling theorem.

The prerequisite is that the signal or its transform in a certain domain is sparse or compressible, and by finding an appropriate sparse transform  $x = \psi\theta$ , most medical images are sparse in some domain.  $\theta$  represents the original image, and  $x$  represents the transformed image. It has been proven that photoacoustic images can be transformed into a sparse domain through various transformations, such as numerical derivatives and wavelet transforms.

In PACT, the objective of reconstruction is to recover the photoacoustic signal  $\theta$  from the measurements  $y$  obtained by the probe, and if the measurement matrix  $K$  obtains  $y$ , then  $y = K\theta$ . Photoacoustic images with CS can typically be reconstructed by solving the constrained optimization problem.

$$\min \|x\|_1 \quad \text{s.t.} \quad y = K\psi^{-1}x. \quad (5)$$

To implement the CS reconstruction of PACT, the discrete expressions of the PACT system in the time and frequency

domains for  $K$  will be given out below. Time domain measurement matrix:

$$K(m, t)_{(i,j)} = \frac{1}{2\pi c} \delta \left( t - \frac{|r_{i,j} - r_m|}{c} \right), \quad (6)$$

$$m = 1, 2, \dots, p; t = s\Delta t, s = 1, 2, \dots, q_s.$$

Frequency-domain reconstruction:

$$K(m, n)_{(i,j)} = i ck_n \frac{\exp(-ik_n|r_{i,j} - r_m|)}{|r_{i,j} - r_m|}, \quad (7)$$

$$m = 1, 2, \dots, p; n = 1, 2, \dots, q_n.$$

Wherein,  $r_{i,j}$  represents the Cartesian coordinates of the image pixels,  $r_m$  represents the position of the probe,  $p$  represents the number of probes,  $q_s$  represents the number of sampling points in the time domain,  $q_n$  represents the number of sampling points in the frequency domain.

**3.4.2 Reconstruction Method.** Based on the above CS theory, the CS-based reconstruction model for PACT is represented by the following formula:

$$\arg \min_x F = \|\Phi x - y\|_2^2 + \alpha \|x\|_1 + \beta \text{TV}(\Psi' x) \quad (8)$$

$F$  is an objective function composed of three parts. The first part represents the squared error between the estimated measurement values of the reconstructed signal and the experimental measurement values. The second part represents the  $\ell_1$  norm of the signal in the sparse domain. The third part represents the total variation (TV) penalty of the signal.  $\alpha$  and  $\beta$  are regularization parameters that determine the trade-off between data consistency and sparsity.

## 4 IMPLEMENTATION

The experimental evaluation contains two distinct segments. The initial segment encompasses a comprehensive assessment of image reconstruction algorithms, specifically juxtaposing back-projection and iterative methodologies. This evaluation rigorously examines the comparative imaging quality and temporal efficiency inherent to each algorithm, thereby elucidating the respective merits and limitations of the time-reversal reconstruction approach via a meticulous comparative analysis.

The subsequent segment delves into the application of GPU parallel computing within the iterative reconstruction framework, as applied to angiographic imaging of murine vasculature. This involves a strategic enhancement of the conventional serial reconstruction paradigm, wherein parallel optimization techniques are employed to significantly expedite the computational process.

The computational hardware employed in this study consists of a CPU model Intel(R) Xeon(R) CPU E5-2620 v2 @2.10GHz, featuring a dual physical GPU setup, with a memory capacity of 64GB, and a substantial storage space of 6TB.

GPU Model	Tesla K40c
CUDA Version	8.0
Compute Capability	3.5
Memory	12 GBytes
Number of CUDA Cores	2880
Constant Memory	65536 bytes
Shared Memory per Block	49152 bytes
Texture Memory	512 bytes

**Table 1: Tesla K40c GPU Configuration**

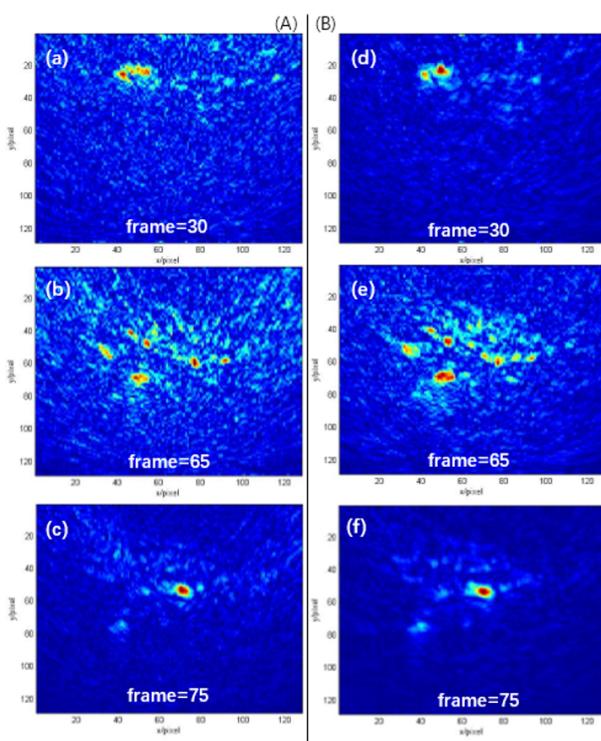
The graphics processing unit (GPU) utilized for the research is the NVIDIA-produced Tesla K40c, built on the NVIDIA Kepler™ architecture, and recognized as a formidable contender in high-performance computing cards. The attributes of the GPU, as empirically determined, are delineated in Table 1. The Tesla K40c GPU boasts a computational capability of 3.5, with a global memory of 12GBytes, a constant memory provision of 65536 bytes, and an allowance of a maximum shared memory of 49152 bytes within each thread block. The GTX960 series permits the configuration of the grid to the maximal dimensions of (2147483647, 65535, 65535), where the maximum number of threads allowed per thread block is set at 1024.

## 4.1 Experiment 1

In this experiment, photoacoustic imaging of murine vascular structures was reconstructed and analyzed. Comparative evaluations were conducted between back-projection algorithms and iterative reconstruction algorithms to ascertain their imaging efficacy and temporal efficiency. This analysis enabled the identification of the strengths and limitations inherent in the iterative approach.

The photoacoustic imaging system captured 166 frames of murine vascular imaging. Due to the large size of the images and consequent slow processing speeds, three frames were selected for the experiment. For each frame, a segment of 127 pixels by 127 pixels was extracted for the reconstruction trials. As illustrated in Figure 9, Section (A) displays the reconstructed images of the 30th, 65th, and 75th frames using the back-projection algorithm. Corresponding images in Section (B) were reconstructed using the iterative algorithm. The reconstruction times are presented in Table 2.

Analysis of Figure 9 reveals that, for the same frame, the iterative algorithm visually outperforms the back-projection algorithm. The latter's suboptimal reconstruction results are due to its non-exact solution-based approach, theoretically possessing inherent shortcomings: back-projection involves distributing weights along an arc, which may lead to non-zero grayscale values in the reconstructed image where the original image had zero values, producing artifacts that blur



**Figure 9: Comparison of filtered back-projection reconstruction and time-reversal reconstruction of mouse vascular photoacoustic images. (A) Mouse vascular photoacoustic image reconstructed with the back-projection algorithm; (B) Mouse vascular photoacoustic image reconstructed with the time-reversal reconstruction algorithm.**

Method	Reconstruction Time (s)		
	frame=30	frame=65	frame=75
BP	2.277	2.712	2.666
IR	118.470	118.814	117.082

**Table 2: Reconstruction times using Back-Projection (BP) and Iterative Reconstruction (IR) methods**

details and exacerbate with the complexity of the imaged object's structure. In contrast, the iterative method is less affected by such imaging artifacts and more accurately restores the original image.

However, as Table 2 indicates, the iterative method has a significant drawback: the computational demand is enormous, and the reconstruction time for a single frame is excessively long. The data show that the time for iterative reconstruction is up to 50 times higher than that for back-projection, greatly limiting its application in photoacoustic

image reconstruction. To enhance the speed of iterative reconstruction in photoacoustic imaging and expand its practical application, it is crucial to apply GPU-based parallel acceleration to the most time-consuming computations.

## 4.2 Experiment 2

**4.2.1 Execution Time Analysis.** In iterative reconstruction, the execution of each step depends on the result of the previous step. This dependency prevents the steps from being processed in parallel. Therefore, serial code is time-tested to identify the most time-consuming execution statements for parallel optimization, thereby achieving an overall optimization effect.

For a frame of 128x128 photoacoustic images, each part of the code was time-tested in MATLAB. The time consumption of the most time-consuming code lines and their respective percentage are shown in Table 3.

During the iterative reconstruction process, a few specific lines of code consume a disproportionately large amount of time, with one line accounting for as much as 96.7% of the total time. The execution time of the program largely depends on the execution rate of this line of code. Therefore, the variables involved in this statement are outputted to clarify their computation process.  $kn$  is a double matrix of size 6144x16384,  $kn'$  is its transpose,  $mk1r$  is a double matrix of size 16384x1, and  $y$  is a double matrix of size 6144x1.

It is evident that in the iterative process, the multiplication of large matrices consumes a significant amount of time, hence the use of CUDA for parallel acceleration of the multiplication process.

### 4.2.2 Parallel Optimization Strategies.

Parallel Optimization Strategies in Program Execution

The optimization of CUDA-based programs fundamentally entails the parallelization of the most computationally intensive tasks to reduce overall execution time. Within the scope of this study, it was observed that matrix multiplication constituted 96.7% of the runtime. Traditional matrix multiplication algorithms implemented in a serial context require three nested loops—wherein the two outer loops calculate the products of corresponding elements across the nth rows and columns, and the innermost loop aggregates these products to establish the definitive value at each specified position within the resultant matrix. The inherent sequentiality of this method necessitates the completion of one computational step before the subsequent one begins, leading to potential task blocking and increased latency in execution.

Addressing these inefficiencies, GPU parallelization techniques have been applied to dissect the computational workload, allocating the calculation of individual elements of the resultant matrix to discrete cores. This concurrent processing paradigm significantly reduces the need for iterative looping

Code	Calls	Total Time(s)	%Time
t1=2*kn** (kn*mklr-y);	90	135.485	96.7%
t1=-2*kn*y;	1	2.060	1.5%
t2=(tv*inv(wmr)*tv*mkl+mk1*(tv*inv(wml)*tv));	90	0.433	0.3%
All other lines		2.177	1.5%

**Table 3: Execution Time Analysis**

and expedites computation, thereby optimizing efficiency. Further, this paper presents an advanced data accumulation strategy that transcends the traditional cyclic approach. A tree-based accumulation technique is proposed to elevate parallelism and diminish temporal overheads. Under this schema, for a given set of  $n$  data points requiring accumulation, the initial phase involves the division of the workload among  $n/2$  CUDA threads, each handling the addition of two data elements in a parallel fashion, resulting in  $n/2$  intermediate sums. Subsequent phases halve the thread count iteratively, each time performing parallel pairwise summations until a singular accumulated value is derived. This innovative algorithm not only reduces iterative loop counts but also mitigates thread underutilization, demonstrating a leap forward in computational efficiency for intensive parallel tasks.

In the parallelized matrix multiplication algorithm, ‘Kernel 1’ is responsible for conducting element-wise multiplications between matrices A and B, with the intermediate products being stored within matrix M. Subsequently, ‘Kernel 2’ undertakes the task of aggregating these products, ultimately rendering the final elements of matrix C. The assignment of tasks to the GPU’s processing cores is orchestrated via block and thread indices (bx, by, tx, ty), which is a convention rooted in CUDA programming. This methodology facilitates the simultaneous execution of operations, leveraging the GPU’s architecture for enhanced computational throughput. It should be noted that the provided pseudocode abstracts from the complexities inherent in actual CUDA development, such as the meticulous synchronization of threads and the strategic allocation of shared memory resources. These aspects, while omitted for brevity in the pseudocode, are crucial for the effective implementation of parallel algorithms on GPU platforms.

#### 4.2.3 Optimizing Memory Utilization in Matrix Operations.

In the advancement of GPU computational capabilities, the efficiency of memory access emerges as a predominant bottleneck impacting program execution times. The latency inherent in accessing global memory far exceeds that of shared or texture memory access. Consequently, program optimization increasingly focuses on maximizing the use of

```

1 // Serial Pseudocode
2 // Input matrices A, B and output matrix C,
3 // all of dimensions i x j
4 Input: A(i, j), B(i, j)
5 Output: C(i, j)
6
7 Main() {
8     // Iterate over the rows of matrices A and C
9     For i = 0 to n-1 {
10         // Iterate over the columns of matrices B and C
11         For j = 0 to n-1 {
12             C(i, j) = 0;
13             For k = 0 to n-1 {
14                 C(i, j) = C(i, j) + A(i, k) * B(k, j);
15             }
16         }
17     }
18 }
```

```

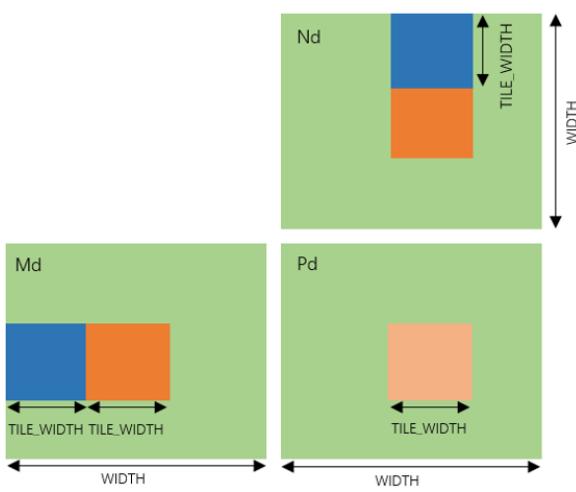
1 // Parallel Pseudocode
2 // Input matrices A, B and output matrix C,
3 // all of dimensions i x j
4
5 Kernel 1:
6 Input: A(i, j), B(i, j)
7 Output: M(i, j)
8
9 1. bx = blockIdx.x, by = blockIdx.y,
10    tx = threadIdx.x, ty = threadIdx.y;
11 2. Set Tiles in the shared memory of the block;
12 3. For i = 0 to n-1 {
13    For j = 0 to n-1 {
14      M(i, j) = A(i, tx) * B(tx, j);
15    }
16  }
17
18 Kernel 2:
19 Input: M(i, j)
20 Output: C(i, j)
21
22 // Accumulate the products in M
23 For k = 0 to n-1 {
24   C(i, j) = C(i, j) + M(i, j);
25 }
```

**Figure 10: Serial and Parallel Pseudocode of the Matrix Multiplication**

shared and texture memory while curtailing dependence on global memory.

To accelerate computational throughput, a judicious strategy employed involves the segmentation of the dataset required for the calculation of specific portions of the resultant matrix, followed by the transference of this segmented data to shared memory. However, the finite capacity of shared memory dictates a need for meticulous data partitioning,

which is exemplified in Figure 11. This data partitioning paradigm is instrumental in refining memory access patterns, which concomitantly enhances the efficiency of GPU-accelerated programs.



**Figure 11: Optimizing Memory Utilization in Matrix Operations**

Within matrix operation contexts, where Nd and Md correspond to the input matrices and Pd to the output matrix, an adept strategy is to partition the extensive matrix into a series of tiles, effectively modularizing the program into several discrete computational stages. Each stage executes by performing accumulative operations on subsets of Md and Nd, thereby progressively assembling Pd. During these operations, data requisite for the computation of an individual tile is systematically migrated from global to shared memory. This stratagem optimizes data locality for each computational stage, significantly reducing the memory access latency and thereby bolstering the overall computational efficacy.

**4.2.4 Experimental Results.** To evaluate the computational advantages of GPUs relative to CPUs in photoacoustic image reconstruction involving matrix multiplication, this study initially performed a detailed analysis of the time performance associated with the matrix multiplication stages of the reconstruction process. The findings are systematically documented in Table 4.

The comparison confirms that the result matrices generated by both the GPU and CPU are identical; however, the GPU's computational speed is approximately six times faster than that of the CPU, indicating a substantial enhancement in performance efficiency.

The reconstruction of three frames was performed by invoking CUDA functions via MATLAB, and the corresponding

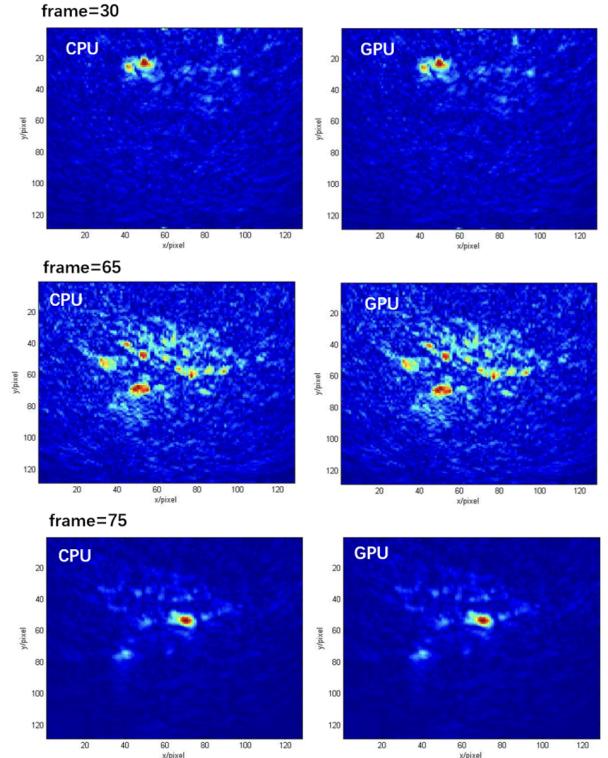
Matrix A	Matrix B	CPU Time (s)	GPU Time (s)
6144x16384	16384x1	42.3	7.8
16384x6144	6144x1	38.4	6.4

**Table 4: Time Consumption Comparison for Matrix Multiplication between GPU and CPU**

Frame	Image Size	CPU Time(s)	GPU Time(s)
30	127*127	118.470	20.062
65	127*127	118.814	20.039
75	127*127	117.082	19.892

**Table 5: Time Consumption Comparison for the reconstruction of 3 frames between GPU and CPU**

run times were meticulously evaluated. The images subjected to these tests are displayed in Figures 12, with the relative processing durations comprehensively detailed in Table 5.



**Figure 12: GPU vs. CPU Imaging Performance Comparison for Frame 30, Frame 65, and Frame 70**

Upon examining the three sets of reconstructed images, it is evident that the outcomes achieved using both the CPU and GPU for image reconstruction are consistent. Data from the table indicates that image reconstruction using solely

the CPU requires approximately 118 seconds. In contrast, employing the GPU to expedite the reconstruction process reduces the duration to about 20 seconds, thereby achieving a speed-up factor of approximately 5.9.

## 5 CONCLUSION

Due to the rapid advancement of GPU hardware and the exponential increase in computing power, the application of GPUs has expanded significantly, bolstered by continual enhancements in GPU programming software. Consequently, GPU parallel computing is emerging as a potent approach for processing large datasets. This paper builds upon this concept, specifically focusing on parallelizing the labor-intensive image reconstruction process in photoacoustic imaging to expedite the imaging workflow.

Through an in-depth examination and exploration of GPU parallel frameworks, this study deepens the understanding of GPU parallel computing mechanisms. It details the conversion of traditional serial code into parallel code and the subsequent optimization of parallel photoacoustic iterative reconstruction code within the CUDA environment to boost program execution efficiency. Experimental results confirm that, in the realm of photoacoustic image reconstruction, GPUs significantly outperform CPUs in terms of image processing efficiency. Nonetheless, additional enhancements are required to fully realize the potential of this technology in clinical applications. Continuous learning and the assimilation of new knowledge are imperative, as ongoing advancements and refinements are expected to enable GPU parallel computing methods to markedly accelerate photoacoustic image reconstruction in the future, thereby promoting its extensive application in fields such as hemodynamics, clinical disease diagnostics, and pharmaceutical development.

Significant opportunities still exist for parallel optimization in the iterative processes of photoacoustic imaging. Future endeavors will aim to further refine these processes, maximizing GPU performance and optimizing data processing speeds.

## REFERENCES

- [1] R Rox Anderson and John A Parrish. 1981. The optics of human skin. *Journal of investigative dermatology* 77, 1 (1981), 13–19.
- [2] Alberto Cano. 2018. A survey on graphic processing unit computing for large-scale data mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8, 1 (2018), e1232.
- [3] CL Philip Chen and Chun-Yang Zhang. 2014. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information sciences* 275 (2014), 314–347.
- [4] Veysel Demir and Atef Z Elsherbeni. 2010. Compute unified device architecture (CUDA) based finite-difference time-domain (FDTD) implementation. *The Applied Computational Electromagnetics Society Journal (ACES)* (2010), 303–314.
- [5] Anders Eklund, Paul Dufort, Daniel Forsberg, and Stephen M LaConte. 2013. Medical image processing on the GPU—Past, present and future. *Medical image analysis* 17, 8 (2013), 1073–1094.
- [6] James G Fujimoto, Costas Pitris, Stephen A Boppart, and Mark E Brezinski. 2000. Optical coherence tomography: an emerging technology for biomedical imaging and optical biopsy. *Neoplasia* 2, 1-2 (2000), 9–25.
- [7] Geng Ku, Xueding Wang, Xueyi Xie, George Stoica, and Lihong V Wang. 2005. Imaging of tumor angiogenesis in rat brains in vivo by photoacoustic tomography. *Applied optics* 44, 5 (2005), 770–775.
- [8] Changhui Li, Andres Aguirre, John Gamelin, Anastasios Maurudis, Quing Zhu, and Lihong V Wang. 2010. Real-time photoacoustic tomography of cortical hemodynamics in small animals. *Journal of biomedical optics* 15, 1 (2010), 010509–010509.
- [9] Christian Lutzweiler and Daniel Razansky. 2013. Optoacoustic imaging and tomography: reconstruction approaches and outstanding challenges in image performance and quantification. *Sensors* 13, 6 (2013), 7345–7384.
- [10] Chris McClanahan. 2010. History and evolution of gpu architecture. *A Survey Paper* 9 (2010), 1–7.
- [11] Jon Peddie. 2023. What is a GPU? In *The History of the GPU-Steps to Invention*. Springer, 333–345.
- [12] Faith M Walker, Charles HD Williamson, Daniel E Sanchez, Colin J Sobek, and Carol L Chambers. 2016. Species from feces: order-wide identification of Chiroptera from guano and other non-invasive genetic samples. *Plos one* 11, 9 (2016), e0162342.
- [13] Xueding Wang, Yongjiang Pang, Geng Ku, Xueyi Xie, George Stoica, and Lihong V Wang. 2003. Noninvasive laser-induced photoacoustic tomography for structural and functional in vivo imaging of the brain. *Nature biotechnology* 21, 7 (2003), 803–806.
- [14] Xi Yang, Yun-Hsuan Chen, Fen Xia, and Mohamad Sawan. 2021. Photoacoustic imaging for monitoring of stroke diseases: A review. *Photoacoustics* 23 (2021), 100287.
- [15] Xinyao Yi, David Stokes, Yonghong Yan, and Chunhua Liao. 2021. CUDAMicroBench: Microbenchmarks to Assist CUDA Performance Programming. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 397–406.
- [16] Xinyao Yi, Anjia Wang, and Yonghong Yan. 2023. NeoRodinia: Evaluation of High-Level Parallel Programming Models and Compiler Transformation for GPU Offloading. (2023).
- [17] Dong-Hee Yoon and Youngsun Han. 2020. Parallel power flow computation trends and applications: A review focusing on GPU. *Energies* 13, 9 (2020), 2147.
- [18] Hao F Zhang, Konstantin Maslov, George Stoica, and Lihong V Wang. 2006. Functional photoacoustic microscopy for high-resolution and noninvasive in vivo imaging. *Nature biotechnology* 24, 7 (2006), 848–851.