A GPU-accelerated Cartesian grid method for PDEs on irregular domain

Liwei Tan[1†], Minsheng Huang[1†], and Wenjun Ying[2,*]

[1] *School of Mathematical Sciences, Shanghai Jiao Tong University, Shanghai 200240, P.R. China.*
[2] *School of Mathematical Sciences, MOE-LSC and Institute of Natural Sciences, Shanghai Jiao Tong University, Minhang, Shanghai 200240, P.R. China.*

Abstract. The kernel-free boundary integral (KFBI) method has successfully solved partial differential equations (PDEs) on irregular domains. Diverging from traditional boundary integral methods, the computation of boundary integrals in KFBI is executed through the resolution of equivalent simple interface problems on Cartesian grids, utilizing fast algorithms. While existing implementations of KFBI methods predominantly utilize CPU platforms, GPU architecture's superior computational capabilities and extensive memory bandwidth offer an efficient resolution to computational bottlenecks. This paper delineates the algorithms adapted for both single-GPU and multiple-GPU applications. On a single GPU, assigning individual threads can control correction, interpolation, and jump calculations. The algorithm is expanded to multiple GPUs to enhance the processing of larger-scale problems. The arrowhead decomposition method is employed in multiple-GPU settings, ensuring optimal computational efficiency and load balancing. Numerical examples show that the proposed algorithm is second-order accurate and efficient. Single-GPU solver speeds 50-200 times than traditional CPU while the eight GPUs distributed solver yields up to 60% parallel efficiency.

AMS subject classifications: 52B10, 65D18, 68U05, 68U07

Key words: Arrowhead decomposition method, GPU-accelerated kernel-free boundary integral method, Irregular domains

1 Introduction

Graphics Processing Units (GPU) are co-processors originally devoted to accelerate graphics processing. In the last years, they are extensively used as massively parallel platforms to run general-purpose programs. This practice is mostly known as General-Purpose

[†] These authors contributed equally to this work.
[*] Corresponding author. *Email addresses:* wying@sjtu.edu.cn (W. Ying)

computing on Graphics Processing Units (GPGPU). This growing trend is confirmed by the number of computers in the top500 ranking that are provided of GPUs, which on November 2023 was 186 [1].

One of the areas taking advantage of the capabilities of this kind of accelerators is scientific computing. There are many recent publications describing works that successfully port code from CPU to GPU, achieving important speedups [2, 3]. Elliptic type problems are widely applied in the fields of electrochemistry [4, 5], electromagnetism [6], computational fluid dynamics [7, 8], shape optimisation problems [9, 10] and other areas in science [11, 12, 13, 14], Solving these problems often requires a substantial computational cost [15].

An effective and accurate approach for solving elliptical equations is the Kernel-Free Boundary Integral (KFBI) method [16, 17, 18], which originates from boundary integral methods. Unlike traditional boundary integral approaches, the KFBI method embeds complex domains into larger, regular computational areas (such as square regions), which are subsequently partitioned using Cartesian grids. The KFBI method not only benefits from the well-conditioning property of the boundary integral equation(BIE) but also avoids explicitly calculating Green's function directly, which is challenging in complex domains [19, 17]. In recent years, the KFBI method has been extensively applied [19, 20, 21, 22, 23].

The substantial memory bandwidth and abundant cores in GPU architecture enable the concurrent execution of thousands of computational tasks, leading to significant acceleration. This renders it an efficient solution for addressing computing bottlenecks. More importantly, the GPU architecture suits the Cartesian grid method since each thread easily controls one grid node. Several related works have addressed the GPU acceleration of Cartesian grid methods in the last ten years [24, 25, 26, 27]: the GPU-accelerated VOF by Rajesh Reddy and R. Banerjee [24], the CUDA-Based IB method by S. K. Layton, A. Krishnan and L. A. Barba [25], the TVD Runge–Kutta method on multiple GPUs by Liang. S, Liu. W and Yuan. L [26], the multiple-GPU based lattice Boltzmann algorithm by Huang. C, Shi. B, He. N and Chai. Z [27].

As a Cartesian grid method, the essential procedure of the KFBI method involves the correction of irregular points and control points on the interface individually, making it inherently well-suited for GPU-accelerated parallel processing. Furthermore, the KFBI method utilizes an FFT-based solver, well-documented in literature for its suitability with GPU or GPU clusters [28, 29, 30, 31, 32], to enhance the efficiency of interface problem computations in iterative procedures. In fact, due to the simple grid topology on Cartesian grids, building a highly parallel GPU-accelerated Cartesian grid solver based on the KFBI method is straightforward. The implementation details of the KFBI solver for a single-GPU version are concisely delineated in section 3, with the corresponding numerical results presented in section 5.

A significant limitation in single-GPU computation is its available memory, which leads to a bottleneck in the size of the computational mesh. In order to expand the calculation scale and improve efficiency, we also study the multiple-GPU architecture in a single node (in one computer), which contains a two-level parallelization: the coarse-grained level composed by GPUs across multiple CPUs at the cost of coordinating GPU-GPU communication via MPI and the fine-grained level formed by CUDA cores on each GPU. Based on the characteristics mentioned above, we have devised a distributed KFBI algorithm that evenly distributes data to each GPU, maximizing the utilization of multiple-GPU parallel capabilities, ensuring computational load balancing, and minimizing inter-GPU communication overhead.

The remainder of the paper is organized as follows. We first introduce the boundary integral method and the KFBI methods in section 2. Section 3 describes implementing the KFBI method on a single GPU. The algorithm is then extended to multiple GPUs and summarised in section 4. The numerical results are presented in section 5. The advantages, limitations, and prospects for the GPU-based KFBI method are discussed in the final section.

2 The kernel-free boundary intergral method

Suppose Ω is a bounded irregular and complex domain in R^2 or R^3 whose boundary $\Gamma = \partial\Omega$ is at least twice continuously differentiable. Let $u(\mathbf{x})$ be an unknown function of $\mathbf{x} \in \mathbf{R}^d (d = 2, or\ 3)$. Assuming $g_D(\mathbf{x})$ and $f(\mathbf{x})$ are known function of $\mathbf{x}$ with sufficient smoothness. $\partial_{\mathbf{n}} u(\mathbf{x})$ denotes the normal derivative of $u(\mathbf{x})$ on the boundary, where $\mathbf{n}$ denotes the unit outward normal on Γ. For simplicity of description, we introduce the KFBI method for the modified Helmholtz equation subject to the Dirichlet boundary condition.

Consider the modified Helmholtz equation

$$\Delta u(\mathbf{x}) - \kappa u(\mathbf{x}) = f(\mathbf{x}), \quad \text{in } \Omega, \tag{2.1}$$

subject to Dirichlet boundary condition

$$u(\mathbf{x}) = g_D(\mathbf{x}), \quad \text{on } \Gamma. \tag{2.2}$$

Here, κ is assumed to be a positive constant for the modified Helmholtz equation in this paper by default.

2.1 Boundary integral equation

As shown in Fig. 1, to solve the boundary value problem above by the KFBI method, we first embed the irregular domain Ω into a larger rectangle domain $\mathcal{B} = \Omega \cup \Omega^c$.

According to the standard BIM [33, 34], let $G(\mathbf{x},\mathbf{y})$ be Green's function on the rectangle $\mathcal{B}$ associated with the elliptic PDE (2.1), which satisfies for $\mathbf{y} \in \mathcal{B}$,

$$\triangle G(\mathbf{x},\mathbf{y}) - \kappa G(\mathbf{x},\mathbf{y}) = \delta(\mathbf{x}-\mathbf{y}), \quad \mathbf{x} \in \mathcal{B}, \tag{2.3}$$

$$G(\mathbf{x},\mathbf{y}) = 0 \quad \mathbf{x} \in \partial\mathcal{B}, \tag{2.4}$$

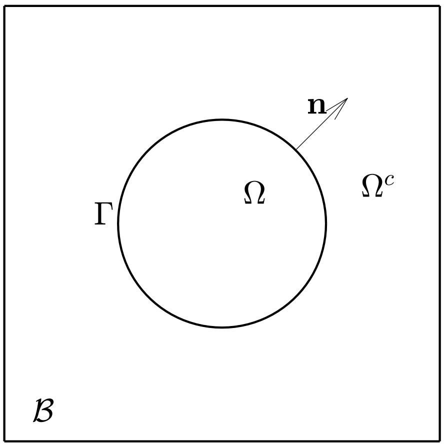


Figure 1: KFBI computation domain

where $\delta(\mathbf{x}-\mathbf{y})$ is the Dirac delta function. Let $\mathbf{n_y}$ be the unit outward normal vector at point $\mathbf{y} \in \Gamma$, and φ be the density function. We first define the double layer boundary integral and volume integral by

$$(W\varphi)(\mathbf{x}) := \int_\Gamma \frac{\partial G(\mathbf{y},\mathbf{x})}{\partial \mathbf{n_y}} \varphi(\mathbf{y}) ds_{\mathbf{y}}, \quad \text{for } \mathbf{x} \in \Omega \cup \Omega^c, \tag{2.5}$$

$$(Yf)(\mathbf{x}) := \int_\Omega G(\mathbf{y},\mathbf{x}) f(\mathbf{y}) d\mathbf{y}, \quad \text{for } \mathbf{x} \in \mathbf{R}^2. \tag{2.6}$$

Thanks to the symbols and properties of the involved potential and volume integral, the Dirichlet BVP (2.1)-(2.2) can be reformulated as a Fredholm boundary integral equation of the second kind [35, 36] by Green's third identity.

$$\frac{1}{2}\varphi(\mathbf{x}) + (W\varphi)(\mathbf{x}) + (Yf)(\mathbf{x}) = g_D(\mathbf{x}), \quad \mathbf{x} \text{ on } \Gamma. \tag{2.7}$$

The solution $u(\mathbf{x})$ to the Dirichlet BVP (2.1)-(2.2) is given by

$$u(\mathbf{x}) = (W\varphi)(\mathbf{x}) + (Yf)(\mathbf{x}), \quad \mathbf{x} \in \Omega. \tag{2.8}$$

Let $M > 2$ be an integer and $\{\mathbf{x}_j\}_{j=0}^{\mathrm{M}}$ be a set of quasi-uniformly spaced points on the domain boundary Γ so that each curve segment $\widetilde{\mathbf{x}_i\mathbf{x}_{i+1}} (i = 0,1,\cdots,M-1)$ has nearly equal length. Numerically, the boundary integral equation (2.7) can be solved by the Richardson iteration: given an initial guess $\varphi_0(\mathbf{x}_m)$, for $k \in \{0,1,2,3,\cdots\}$,$m \in \{0,1,2,\cdots,M\}$, do as follows

$$u_k^+(\mathbf{x}_m) = \frac{1}{2}\varphi_k(\mathbf{x}_m) + (W\varphi_k)(\mathbf{x}_m), \quad \mathbf{x}_m \in \Gamma, \tag{2.9}$$

$$\varphi_{k+1}(\mathbf{x}_m) = \varphi_k(\mathbf{x}_m) + \gamma[\hat{g}_D(\mathbf{x}_m) - u_k^+(\mathbf{x}_m)]. \quad \mathbf{x}_m \in \Gamma. \tag{2.10}$$

Here $\hat{g}_D(\mathbf{x}_m) = g_D(\mathbf{x}_m) - (Yf)(\mathbf{x}_m)$, which only need to calculate once before Richardson iteration. $\mathbf{x}_m$ is a control node located on the boundary Γ. It can be shown that the Richardson iteration is convergent when $\gamma \in (0,1]$. The superscript "+" in the BIE means one-sided limit from the domain Ω. More specifically, let $w(\mathbf{x})$ be an arbitrary piecewise smooth function with discontinuities only existing at the interface Γ. We denote

$$w^+(\mathbf{x}) = \lim_{z \longrightarrow x, z \in \Omega} w(\mathbf{z}). \tag{2.11}$$

similarly, the restriction of $w(\mathbf{x})$ in $\bar{\Omega}^c = R^d \backslash \bar{\Omega}$, $w^-(\mathbf{x})$ is defined as

$$w^-(\mathbf{x}) = \lim_{z \longrightarrow x, z \in \bar{\Omega}^c} w(\mathbf{z}). \tag{2.12}$$

Once the unknown density function $\varphi(\mathbf{x})$ is obtained when the iteration (2.10) converges. The unknown function $u(\mathbf{x})$ can be calculated according to the formula (2.8).

2.2 Indirect evaluation of integrals

In the traditional BIM method, the expression of Green's function must be explicitly known. However, the exact form of Green's function varies with the PDE, boundary condition and the domain. Although Green's function can be replaced with a neural network [37] and has good numerical results in solving Laplace's and Helmholtz's equations, this method currently cannot solve the problem with variable coefficients. Within the framework of the KFBI method, there is no need to know Green's function. The integrals in (2.5)-(2.6) are indirectly evaluated by the equivalent interface problems. In detail, the double layer boundary integral $(W\varphi_k)(\mathbf{x})$ and volume integral $(Yf)(\mathbf{x})$ can be written into the same form:

$$\begin{aligned} &\Delta v(\mathbf{x}) - \kappa v(\mathbf{x}) = \mathcal{F}(\mathbf{x}), && \mathbf{x} \text{ in } \Omega \cup \Omega^c, \\ &[v(\mathbf{x})] = \Phi(\mathbf{x}), && \mathbf{x} \text{ on } \Gamma, \\ &[\partial_{\mathbf{n}} v(\mathbf{x})] = 0, && \mathbf{x} \text{ on } \Gamma, \\ &v(\mathbf{x}) = 0, && \mathbf{x} \text{ on } \partial\mathcal{B}. \end{aligned} \tag{2.13}$$

Here, $[v(\mathbf{x})]$ and $[\partial_{\mathbf{n}} v(\mathbf{x})]$ represent the jumps of unknown $[v(\mathbf{x})] = v^+(\mathbf{x}) - v^-(\mathbf{x})$ and its normal derivatives $[\partial_{\mathbf{n}} v(\mathbf{x})] = \partial_{\mathbf{n}} v^+(\mathbf{x}) - \partial_{\mathbf{n}} v^-(\mathbf{x})$ respectively, $\tilde{f}(\mathbf{x})$ is the zero extension of given function $f(\mathbf{x})$.

Integral	$\mathcal{F}$	Φ
$W\varphi$	$\mathcal{F}=0$	$\Phi=\varphi$
Yf	$\mathcal{F}=\tilde{f}(\mathbf{x})=\begin{cases} f(\mathbf{x}) & \text{in }\Omega \\ 0 & \text{in }\Omega^c \end{cases}$	$\Phi=0$

During the discretization of the interface problem, the discrete linear system of the interface problem (2.13) has to be corrected at the irregular nodes due to the presence of the interface Γ. The correction process needs to calculate jumps, such as $[v]$, $[v_x]$, $[v_y]$, $[v_{xx}]$, $[v_{xy}]$, $[v_{yy}]$ for second-order discretization, as well as modifying function values, as described in section 3.1.

The jumps calculated above not only requires a correction for the discrete system in (2.13), but also interpolation of the grid-based solution $\mathbf{v}_h$ on the boundary. In summary, the second-order finite-difference method for solving interface is described in Algorithm 1:

```
Algorithm 1 Second-order finite difference method for interface problem (2.13)
1: Initialize the Cartesian grid of bounded box $\mathcal{B}$.
2: Partition the interface $\Gamma$ by quasi-uniformly control points.
3: Discretize the interface problem (2.13) with second-order finite difference method.
4: Compute jumps and correct the $\tilde{f}(\mathbf{x})$ at the irregular nodes.
5: Solve the linear system by FFT-based or geometric multigrid fast solvers.
6: Interpolate the solution to get one-sided boundary value.
```

The first step is to partition box $\mathcal{B}$ into Cartesian grid and divide the Cartesian grid nodes into regular and irregular nodes according to the boundary location. As shown in Fig.2(b), we define the irregular points if some of their adjacent grid nodes go cross the boundary. Black squares denote the interior irregular nodes while blue triangles denote exterior. Others are regular nodes.

For procedure of implementing steps 2-6 in Algorithm 1 on the CPU, we recommend reading reference [16] for detail. In following section, we will focus on explaining how to execute algorithms 3-6 on single GPU in section 3 and multiple GPUs in section 4.

3 Single GPU algorithm

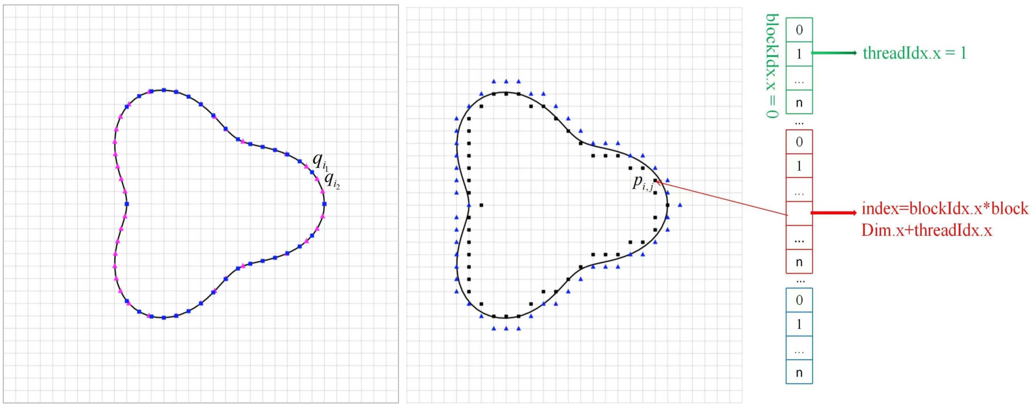
3.1 Correction

The KFBI method involves making corrections to each irregular node [16]. Therefore, on a single GPU, each thread block is comprised of 1,024 threads, and each thread corresponds to precisely one irregular node for correction.

Suppose that the rectangle domain $\mathcal{B}$ is partitioned into a uniform Cartesian grid with nodes $\{\mathbf{p}_{i,j} := (x_i,y_j) : 0 \le i \le I, 0 \le j \le J\}$. For simplicity, assume the grid has the same spacing in the horizontal and vertical directions and denote by $h > 0$ the spacing parameter, i.e., $h = x_{i+1} - x_i = y_{j+1} - y_j$. Let $v_{i,j} = v_h(\mathbf{p}_{i,j})$ be the finite difference approximation of $v(\mathbf{x})$ at the grid node $\mathbf{p}_{i,j}$. One can describe the GPU version of the correction method as the following Algorithm 2 and schematic plot in Fig. 2.

Algorithm 2 Correction Procedure

```
Input: intersection nodes, irregular nodes, $\Phi$,$\mathcal{F}$.
Output: corrected right hand side $\tilde{f}_{i,j}$ on each irregular nodes.
Locate index of irregular nodes by index ← blockIdx.x × blockDim.x + blockIdx.x
For irregular nodes $\mathbf{p}_{i,j}$, find the corresponding intersection nodes set $\mathcal{Q} = \{\mathbf{q}_{i_1},\mathbf{q}_{i_2},\cdots\}$
for each $\mathbf{q}_{i_k}$ in $\mathcal{Q}$ do
    Interpolate $\Phi$ and calculate jumps of derivatives at of $\mathbf{q}_{i_k}$.
    Evaluate and modify correction value $\tilde{f}_{i,j}$ by the discrete scheme on irregular nodes $\mathbf{p}_{i,j}$.
end for
```



(a) Intersection nodes

(b) Irregular nodes

Figure 2: A graphical scheme for intersection nodes 2(a) and irregular nodes 2(b). In the left, pink triangles denote intersection nodes with x direction while blue squares refer to intersection nodes with y direction. In the right, each irregular node is controlled by one thread. Blue triangle denotes exterior irregular node while black denotes inner node.

3.2 Interpolation

Fig.3 shows the six-point stencil of interpolated point $\mathbf{z}_k$ located in the shadow region of a grid cell. The stencil consists of 6 points and can be obtained by rotation or reflection transformation if $\mathbf{z}_k$ in another grid cell. As shown in Fig.3, $\mathbf{p}_{i,j}$ are the grid points for extracting value at a point $\mathbf{z}_k \in \Gamma$. The corresponding algorithm can be described in Algorithm 3:

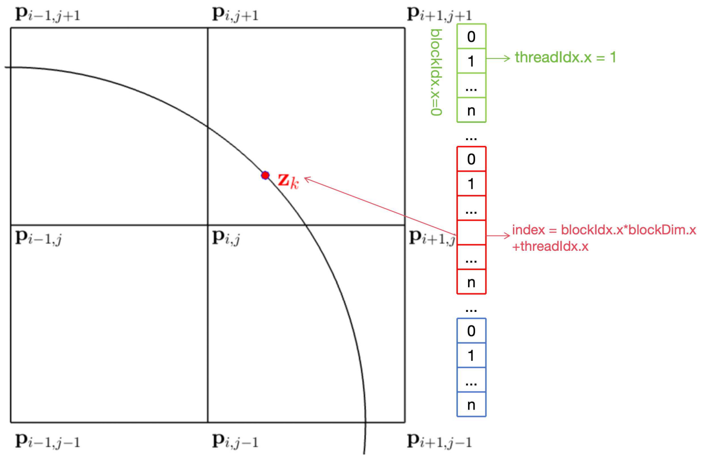


Figure 3: Interpolation schematic diagram. The evaluation of the control node z_k depends on neighbor grid value $\mathbf{p}_{I,J}, I = \{i-1, i, i+1\}, J = \{j-1, j, j+1\}$. For parallelization, every control node is controlled by one thread.

Algorithm 3 Interpolation Procedure

```
Input: intersection nodes, irregular nodes, $\Phi, \mathcal{F}$.
Output: the value and its derivatives at control node $\mathbf{z}_k$.
Locate index of control nodes by index $\leftarrow$ blockIdx.x $\times$ blockDim.x + blockIdx.x.
for control node $\mathbf{z}_k \in \Gamma$: do
    Compute corresponding jumps of value and its derivatives respectively.
    Find interpolate stencil and formulate interpolate linear system.
    Solve linear system and extract the boundary data on $\mathbf{z}_k$.
end for
```

3.3 GMRES iteration with FFT-based solver

• **FFT-based solver** The most important feature of the KFBI method is the conversion of a volumn or boundary integral into an interface problem. The solution to the interface problem consists of two steps. The first step is to make corrections to ensure accuracy, and the second is to solve it using a fast algorithm, such as FFT-based solver [38]. The discrete linear system can be solved using the FFT-based solver on GPU by implementing CUDA programming and invoking cusparse [39] and cufft libraries [40]. The algorithm can be described as Algorithm 4:

Algorithm 4 FFT-based solver of modified Helmholtz equation in GPU

```
Input: corrected value $\tilde{f}_{i,j}$.
Output: the solution of interface problem: $v_{i,j}$.
Take the FFT transform on the $\tilde{f}_{i,j}$ on $y$-dimension and get transformed $\hat{\tilde{f}}_{i,j}$.
Solve tridiagonal linear system with right hand side term $\hat{\tilde{f}}_{i,j}$, and get solution $\hat{v}_{i,j}$.
Take the inverse FFT transform on $\hat{v}_{i,j}$ on $y$-dimension, and get final solution $v_{i,j}$.
```

In Algorithm 4, it should be pointed out that FFT transforms depend on the boundary condition on $\partial\mathcal{B}$. If $\partial\mathcal{B}$ is the periodic, Dirichlet or Neumann, one needs to do periodic, Fast Sine Transform(FST), or Fast Cosine Transform(FCT), respectively. There is no difference in the influence of the results in the three scenarios. However, in this paper, we perform FFT on one dimension and solve the tridiagonal resulting system on another, which is the fastest and most efficient.

• **GMRES iteration** GMRES is an iterative method for solving nonsymmetric linear systems. The method aims to approximate the solution by the vector in a Krylov subspace with minimal residual. The condition number of BIE (2.7) is relatively small, allowing for fast convergence of GMRES iteration [41].

From (2.9), we denote :

$$\mathcal{K}_D(\varphi)(\mathbf{x}) := (\frac{1}{2}I+W)(\varphi)(\mathbf{x}), \quad \mathbf{x} \in \Gamma \tag{3.1}$$

The main points of the GMRES method with restarts are presented in Algorithm 5. It is worth noting that the matrix-vector product in GMRES is replaced by the KFBI method(line 2 and 6). As for computing vector norm(line 3, 11, and 17), inner product(line 8), scalar-vector operation(line 4 and 12), and some axpy operation(line 9), they are implemented by the cuda kernel functions. Since the calculated φ need to be reintegrated into the next iteration, the GPU information needs to be transferred back to the CPU after each GMRES iteration to obtain updated M_{m-1} to M_m(line 14). Therefore, the calculations for the least squares method are also performed on the CPU(line 15).

Algorithm 5 GMRES with restarts in GPU

Input: corrected value $\tilde{f}_{i,j}$.

Output: the solution of BIE(2.7): φ_m.

```
convergence = false
while convergence == false do
    $r_0 = \hat{g}_D - \mathcal{K}_D \varphi$
    $\beta = ||r_0||_2$
    $\mu_1 = \frac{r_0}{\beta}$
    for $j = 1$ to $m$ do
        $w_j = \mathcal{K}_D \mu_j$
        for $i = 1$ to $j$ do
            $h_{i,j} = (w_j, \mu_i)$
            $w_j = w_j - h_{i,j} \mu_i$
        end for
        $h_{j+1,j} = ||w_j||_2$
        $\mu_{j+1} = \frac{w_j}{h_{j+1,j}}$
    end for
    Set $M_m = [\mu_1, \cdots, \mu_m]$, and $\hat{H}_m = (h_{i,j})$ is upper Hessenberg matrix of order $(m+1) \times m$
    Solve a least-square problem: $\min_{y \in \mathbf{R}^2} ||\beta \mathbf{e}_1 - \hat{H}_m y||_2$
    $\varphi_m = \varphi_0 + M_m y_m$
    if $||\hat{g}_D - \mathcal{K} \varphi_m||_2 < \epsilon$ then
        convergence = true
    end if
    $\varphi_0 = \varphi_m$
end while
```

4 Multi-GPUs algorithm

4.1 Domain decomposition

While a single GPU has demonstrated commendable performance in numerous applications, the computational demands entailed in simulating extensive more large-scale problems surpass the capabilities of a single GPU. It is necessary to explore the development of parallelization techniques using multiple GPUs to address this issue. For descriptive convenience, our multiple-GPU algorithm is presented using a two-dimensional grid as a paradigm, with the situation for three-dimensional grids being analogous.

Our study employs the domain decomposition method to partition a 2D grid with $Nx \times Ny$ dimensions into m parts along the $x-$coordinate direction. Each part represents a sub-domain; the total number of sub-domains is denoted as m,m equals the total number of GPUs in use. Each sub-domain is assigned to a corresponding process equipped with a GPU for computation. Throughout the simulation, the variables of each sub-domain persistently reside in the global memory of the assigned GPU.

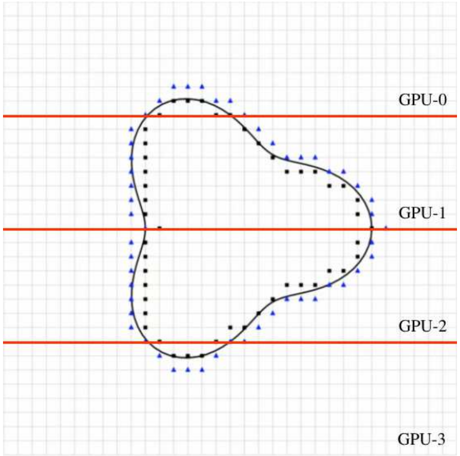


Figure 4: Partition the Cartesian grid into four subdomains along the y-axis, with each subdomain assigned to a dedicated process equipped with a GPU for computation.

In the interface problem-solving process, each process stores the relevant information of curve points within its allocated region. Initially, all the data points on the boundary are gathered to calculate the solutions at these specific curve points. Subsequently, the computed results are distributed back to the respective processes.

Before set up the iteration in Section 3.3, the CPU's grid data and boundary data are distributed to their respective processes based on regions. Subsequently, this data is efficiently copied to the device. Throughout the iterative solving process, each process is responsible for processing the pertinent information related to grid points, control points, and intersection nodes within its designated region, and all computations are carried out on the GPU. Upon completion of the iteration, the processed data is then returned to the host system. This approach ensures a standardized and optimized procedure for utilizing CUDA-enabled devices to expedite the solving process.

4.2 Data communication

In order to facilitate efficient data exchange, MPI is utilized on the CPU to transfer data from two layers of internal units adjacent to subdomain boundaries. We allocate memory spaces on both the host and the device to store secondary boundary data. Once the boundary of a specific subdomain is computed, each device uploads the necessary boundary data to the virtual region, which is then downloaded by neighbouring devices to prepare for the subsequent computational steps. This approach optimizes data exchange while minimizing data redundancy between the CPU and GPU.

Given that there are fewer control points than grid points, the time and computational cost associated with collecting and disseminating potential values, denoted as ϕ and ψ,

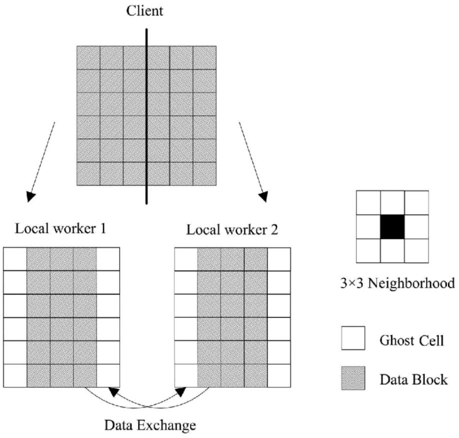


Figure 5: Each process is accompanied by a peripheral layer of a virtual subdomain, situated at the boundaries of their respective regions, which serves the purpose of receiving boundary data transmitted by neighboring processes.

across various boundary regions is relatively low. During the process of solving the interface problem (2.13), each process is tasked with obtaining potential function information for all control points. To streamline this procedure, MPI is employed to consolidate the potential function information before the interface problem calculation begins. Upon completion of the calculation, MPI is once again used to distribute the results back to their respective processes.

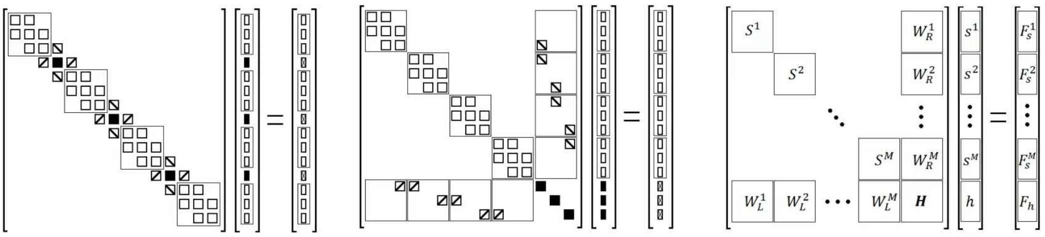
4.3 Poisson solver

For the FFT-based solver of the Poisson equation, we still follow the process Algorithm 4. Unlike single GPU, we solve tridiagonal linear equations using the distributed arrowhead decomposition method(ADM) [42]. The ADM is an efficient algorithm, and the resulting system is suited for designing distributed algorithms for each sub-domain on the corresponding GPU.

4.3.1 Arrowhead decomposition method

The linear equation system to be solved is denoted as : $Au = f$. Fig.6 depicts the concept of ADM. A similarity transformation transforms the initial block-tridiagonal linear system(4.1) into an equivalent block matrix form. This reorganization is particularly advantageous for region decomposition and the design of distributed algorithms, as each block matrix's linear system exhibits a degree of independence. The reordering is carried out by exchanging block rows and block columns, which, in turn, affects the elements of the unknown vector and the right-hand side vector. The resulting matrix structure can be represented as a 2×2 block matrix.

$$\begin{pmatrix} \mathbf{S} & \mathbf{W}_R \\ \mathbf{W}_L & \mathbf{H} \end{pmatrix} \begin{pmatrix} s \\ h \end{pmatrix} = \begin{pmatrix} F_s \\ F_h \end{pmatrix} \tag{4.1}$$



(a) Intial linear system (b) Rearranged linear system (c) Notation of the rearranged system

Figure 6: A graphical scheme for rearrangement of the initial block-tridiagonal linear system into the equivalent form, coming from [42]. Left: the initial matrix A with blocks. Center: after rearrangement, the initial matrix becomes "arrowhead" matrix. Right: the denotation of the "arrowhead" matrix.

In this context, the unknown solution vector h corresponds to the movable component of the complete solution, as illustrated in Figure 6. The square super-block $\mathbf{S}$ comprises new independent tridiagonal blocks S^k, with $k = 1,\ldots,M$, positioned along the diagonal. The matrix elements $\mathbf{H}$ form the lower-right coupled super-block, representing the coupling of unknowns at the interface. Additionally, the other horizontal super-blocks $\mathbf{W}_R$ and $\mathbf{W}_L$ are supplementary components within the matrix, signifying the internal unknowns of the coupling processors at the interface. The following relationships determine the solution of the system(4.1).

$$\begin{cases} s = \mathbf{S}^{-1}F_s - \mathbf{S}^{-1}\mathbf{W}_R h \\ h = \left(\mathbf{H} - \mathbf{W}_L\mathbf{S}^{-1}\mathbf{W}_R\right)^{-1}\left(F_h - \mathbf{W}_L\mathbf{S}^{-1}F_s\right) \end{cases} \tag{4.2}$$

These relationships involve matrix products and inversions, which can be parallelized to a certain extent. The independence of blocks S^k allows for distributed parallel computation of the products $\mathbf{S}^{-1}F_s$. In practice, rather than computing inversions, we efficiently solve the distributed linear systems $Sx = F_s$ due to the special properties of S^k. The sparse structure of $\mathbf{W}_L$ and $\mathbf{W}_R$ significantly reduces the number of matrix operations in(4.2). Once a portion of the total solution $\mathbf{X} = (s,h)^T$ is obtained from the second relationship in(4.2), i.e., $h = \left(h^1,\ldots,h^{M-1}\right)^T$, the remaining parts can be computed in parallel over GPU k using the provided formula.

$$s^k = z^k - Z^k h^{k-1} - Z^k h^k \tag{4.3}$$

where formally $z^k = \left(S^k\right)^{-1}F_s, Z^k = \left(S^k\right)^{-1}W_R^k$, and $h^0 = h^M = 0$.

The distributed algorithm for solving tridiagonal matrices can be outlined as follows:

Algorithm 6 Distributed solver for arrowhead decomposition method

```
Input: corrected value $\tilde{f}_{i,j}$ and known matrix $A$.
Output: the solution of $Au = f$.
1: Decompose the coefficient matrix $A$ and the right-hand side vector $f$ into $m$ subregions, where $m$ equals the number of processes.
2: Precompute the Schur complement matrices $(H-W_LS^{-1}W_R)^{-1}$, $S^{-1}W_R$ independently.
3: Compute $S_k^{-1}F_s$, $k = \{1,2,\cdots,m\}$ and exchange data by passing the first row of $x_k$ from the $(k+1)^{th}$ processes to the auxiliary boundary of the $k^{th}$ process to compute $\big(F_h-\mathbf{W}_L\mathbf{S}^{-1}F_s\big)$.
4: Compute $h^k,k = \{1,\cdots,m-1\}$ by (4.1) and $h^0,h^m$ are set to 0.
5: Evaluate $s^k,k = \{1,\cdots,m\}$ by formula (4.3), where the vector $h^{k-1}$ is passed from $(k-1)^{th}$ process to the auxiliary boundary of the process $k$.
```

In step 1, each process is assigned the task of handling storage and computations for variables within its respective subregion. This approach ensures a standardized and optimized procedure for distributing the workload across multiple processes. In step 2, computing these matrices avoid redundant calculation during the iterations. For the vector update operations in both step 1 and 2, each vector is divided into some segments according to the number of devices. Each segment pair forms a subtask in a device and these subtasks are computed simultaneously. For the dot product in point, the vectors $\vec{x}$ and $\vec{y}$ are cut into segments and computes on the devices in parallel firstly, then the local sum was calculated using the API *reduce* in the thrust library. The MPI is used to solve the final global sum, the vector norm can easily be calculated when the dot product is acquired. In our program, an original vector is partitioned sequentially. Any vector is stored as a one-dimensional array. A set of vectors are managed through a two-dimensional array. For the matrix-vectors product, here is actually the solution to the interface problem.

4.4 Algorthm summary

The structure of the GPU-accelerated distributed KFBI algorithm can be found hereinafter. The individual steps are interleaved by communication calls, as visualized by printing the communication in italic.

- Procedure 1: Initialize the Cartesian grid

  1. Use quasi-uniformly spaced points $\mathbf{z}_i$ to discretize the interface Γ;
  2. Partition $\mathcal{B}$ into a uniform Cartesian grid $\mathcal{T}_h$;
  3. Identify the regular and irregular nodes of the Cartesian grid;
  4. Find intersecting points located between Γ and Cartesian grid line.

- Procedure 2: Evaluate boundary data on control nodes

  1. The boundary point data is scattered to the corresponding process according to the region;
  2. Compute jumps of partial derivatives at control nodes;
  3. Solve interface problems by section 4.3;
  4. Exchange grid data between adjacent processes;
  5. Extract boundary data $u^{+}(\mathbf{x})$ or $\partial_{\mathbf{n}} u(\mathbf{x})$ for Dirichlet or Neumann BVP respectively;
  6. Collect boundary point data and calculate errors.

- Procedure 3: The GMRES iteration

  1. Choose an initial guess φ_0 or ψ_0 and distribute it to different GPUs to start the GMRES iteration for the Dirichlet BVP or Neumann BVP respectively;
  2. Perform the GMRES iteration according to 3.3;
  3. Repeat the previous steps 2 until the discrete residual of the boundary integral equation is sufficiently small in some norm.

5 Numerical Results

To study the numerical accuracy and efficiency of the methods above, in this section, we present the numerical results for the Laplace equations, the reaction-diffusion equations, and the Stokes equations in an irregular domain. The bounding box $\mathcal{B}$ embedding the domain Ω for solving the interface problem is specified as a square(cube), whose size as well as the curve(surface) parameters are given respectively in the description of each numerical example.

The following examples give the convergent error of the numerical discretization scheme. Taking two dimensions as an example, the error is defined as e_{ij} with $e_{ij} = (u_h)_{ij} - (u^*)_{ij}$, where N is the number of interior grid nodes, u^* is the exact solution, u_h is the numerical solution with step size h. Denote by $\|\mathrm{e}_h\|_\infty$ and $\|\mathrm{e}_h\|_2$ the discrete maximum norm and the scaled discrete l_2-norm of e_{ij} respectively, i.e.,

$$\|\mathbf{e}_h\|_\infty = \max_{(x_i,y_j)\in\Omega} |e_{ij}|$$

$$\|\mathbf{e}_h\|_2 = \sqrt{\frac{1}{N}\sum_{(x_i,y_j)\in\Omega} |e_{ij}|^2}$$

To check the algorithm's accuracy, we verify the numerical error in each case with the grid refinement. The GMRES iteration stops when the iterated residual in the discrete

ℓ^2-norm relative to that of the initial residual is less than a prescribed tolerance and is fixed to be 10^{-8}. The corresponding table for each case lists the step size, the number of grid points, the CPU times, the GPU times, and the speedup ratios. Numerical results on the Cartesian grid to the problem are also displayed in the plots for each fixed time.

In addition, we perform numerical experiments on eight NVIDIA GeForce RTX 3090 graphics cards, which contain 10496 cores organized in 84 streaming multiprocessors (MPs). Moreover, it provides 24GB of device memory with a memory bandwidth of 936GB/s, accessible by all its cores and the CPU through the Intel(R) Xeon(R) Gold 6330 CPU with 28 cores.

Poisson	grid size	512^2	1024^2	2048^2	4096^2
FFT	GPU time	0.19 s	0.27 s	0.66 s	1.65 s
Multigrid	GPU time	0.25 s	0.61 s	2.08 s	8.66 s

Table 1: Comparison of different Poisson solvers on DBVP of the Laplace equation.

Iteration	grid size	512^2	1024^2	2048^2	4096^2
Richardson	CPU time	5.62 s	23.11 s	92.05 s	380.12 s
	GPU time	0.19 s	0.27 s	0.66 s	1.65 s
GMRES	CPU time	1.38 s	5.81 s	25.92 s	110.21 s
	GPU time	0.13 s	0.16 s	0.18 s	0.25 s
BiCGSTAB	CPU time	1.99 s	8.01 s	34.47 s	147.49 s
	GPU time	0.45 s	0.45 s	0.63 s	1.01 s

Table 2: Comparison of various iterative methods for solving the Dirichlet boundary value problem (DBVP) associated with the Laplace equation, with a fixed tolerance level of $1e-08$ for the Richardson, GMRES, and BiCGSTAB methods.

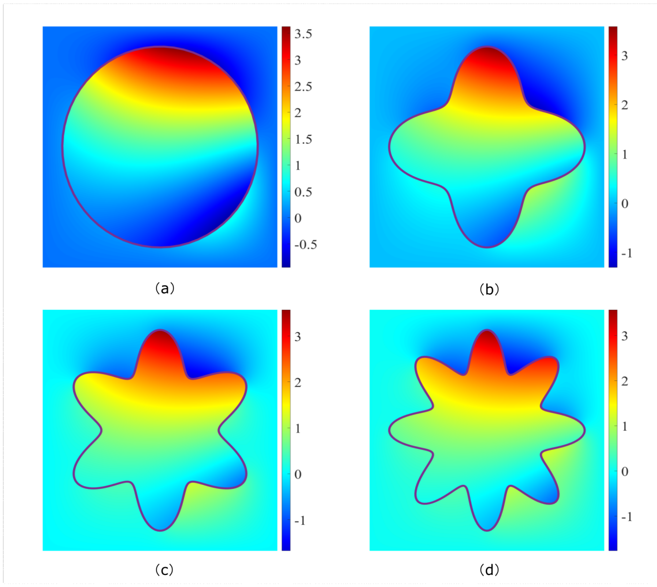
5.1 Single GPU results

Example 1. The results presented in Tab. 1 clearly demonstrate that, within the specified range of simulation scales, the FFT+tridiagonal Poisson solver outperforms the geometric multigrid solver in terms of efficiency. This observation is consistent with the findings reported in [43]. Analyzing the results from Tab. 2, it is evident that the GMRES method achieves the lowest iteration number, resulting in reduced CPU and GPU time consumption. As a result, for subsequent numerical experiments, this study adopts the parallel FFT+tridiagonal Poisson solver in combination with the GMRES method for computational purposes.

This example solves the boundary value problem of the Laplace equation on the circle domain(the parameters $r_a = 1.0$, $r_b = 1.0$) and the rotated star-shaped domain(the parameters $m = 4.0,6.0,8.0$, $r = 1.0$, $c = 0.2$), with the Dirichlet boundary condition. The boundary conditions are chosen so that the exact solution reads

$$u(x,y) = \exp(x)\cos(y) + \exp(y)\sin(x)$$

The bounding box $\mathcal{B}$ for the interface problem is set to be $\mathcal{B}=(-1.2,1.2)\times(-1.2,1.2)$. Numerical results are plotted in Fig. 7.



(a)

(b)

(c)

(d)

Figure 7: The numerical solutions on the circle and star-shaped domain on the 2048×2048 grid.(a)Circle domain. The radius r = 1.0. (b)The star-shaped domain. The fold number m = 4.0, radius r = 1.0, c = 0.2. (c)The star-shaped domain. The fold number m = 6.0, radius r = 1.0, c = 0.2. (d)The star-shaped domain. The fold number m = 8.0, radius r = 1.0, c = 0.2.

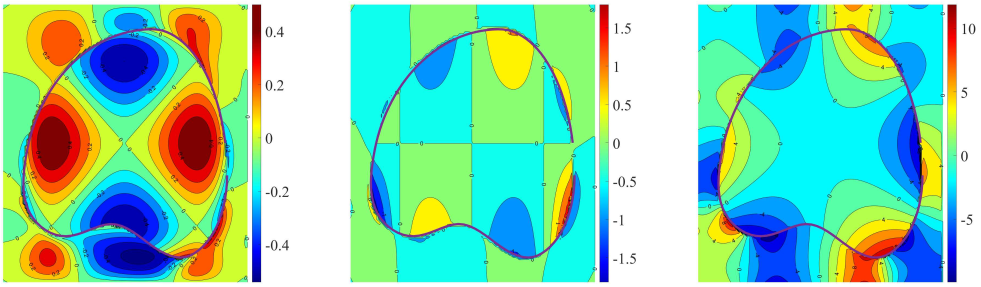
Example 2. This example solves the boundary value problem of the Stokes equation on the heart-shaped domain, with the Dirichlet boundary condition. A Cartesian grid-based MAC Scheme is applied to solve the Stokes equation. This approach places the pressure p at the cell center, the $x-$component velocity $u^{(1)}$ and the $y-$component velocity $u^{(2)}$ at the midpoints of the vertical and horizontal edges of each cell, respectively. The method is detailed in [44]. The boundary conditions are chosen so that the exact solution reads

$$\begin{aligned} u^{(1)}(x,y) &= x(x^2-3y^2)+1.5(1-(x^2+y^2))x \\ u^{(2)}(x,y) &= -1.5(1-(x^2+y^2)^2)y \\ p(x,y) &= 6(y^2-x^2) \end{aligned} \tag{5.1}$$

The bounding box $\mathcal{B}$ for the interface problem is set to be $\mathcal{B}=(-1.2,1.2)\times(-1.2,1.2)$. The execution time on the CPU and GPU are summarized in Tab. 3. Numerical results are plotted in Fig. 8.

Boundary	grid size	64^2	128^2	256^2	512^2
Dirichlet	CPU time	6.95 s	31.27 s	135.26 s	551.24 s
	GPU time	1.36 s	1.85 s	3.25 s	5.51 s
	Speedup	5.11	16.90	41.61	100.04

Table 3: BVP of the Stokes equation on the heart-shaped domain.



(a) The velocity field $u^{(1)}$ (b) The velocity field $u^{(2)}$ (c) The pressure field p

Figure 8: The numerical solutions for example 2 on the 512×512 grid.

Example 3. This example solves the Gray-Scott model which consists of two singularly perturbed reaction-diffusion equations given by

$$u_t = \epsilon_1 \Delta u + \frac{1}{\epsilon_0}\left[\gamma(1-u) - uv^2\right]$$
$$v_t = \epsilon_2 \Delta v + \frac{1}{\epsilon_0}\left[uv^2 - (\gamma+\kappa)v\right]$$

Here, $u = u(x,y,t)$ and $v = v(x,y,t)$ are two unknown smooth functions, describing the concentration of some chemical substances in a bounded domain Ω for $t \geqslant 0;\gamma$ and κ are respectively the feed and removal rate; ϵ_0,ϵ_1 and ϵ_2 are small reactive or diffusive coefficients. In this example, the model is assumed to satisfy the homogeneous Neumann boundary condition that $\partial_n u = \partial_n v = 0$ on $\partial\Omega$, initial condition and the involved parameters are specified as follows

$$v(x,y,0) = \begin{cases} \frac{1}{4}\sin^2(4\pi x)\sin^2(4\pi y), & -0.25 \leqslant x,y \leqslant 0.25 \\ 0, & \text{others.} \end{cases}$$
$$u(x,y,0) = 1 - 2v(x,y,0)$$
$$\gamma = 0.024, \kappa = 0.06, \epsilon_0 = 0.01, \epsilon_1 = 0.008, \epsilon_2 = 0.004$$

The bounding box $\mathcal{B}$ for the interface problem is set to be $\mathcal{B} = (-2.0,2.0)\times(-2.0,2.0)$ and the tolerance is 10^{-8}. Time direction is discretized by the second-order Strang splitting method [45]. The numerical results when $T = 1,2,7,11,17,21,25,50$ are plotted in

Fig. 9. Tab. 4 we present the execution time on the CPU and GPU of the parallel algorithm for different computing scales, In order to verify the computational efficiency and stability, the GPU acceleration ratio and numerical accuracy are also shown in the table. It is calculated selecting four different problem sizes: 128×128, 256×256, 512×512, 1024×1024, the time step is increasing with the increase of the grid size. From the table we can see that the GPU acceleration ratio increases with increasing of the computation scale, It can be seen that a better performance is obtained when large problems are considered, which means our parallel method scales well.

boundary condition	grid size	128×128	256×256	512×512	1024×1024
	Time steps	8	16	32	64
Neumann	CPU time	1.21 s	8.98 s	74.97 s	633.78 s
	GPU time	0.40 s	1.04 s	3.29 s	12.07 s
	Speedup	3.0250	8.6346	22.7872	52.5086

Table 4: Simulation time of CPU-based and GPU-based KFBI method, as well as the speedup achieved by GPU-based solver over the CPU-based solver.

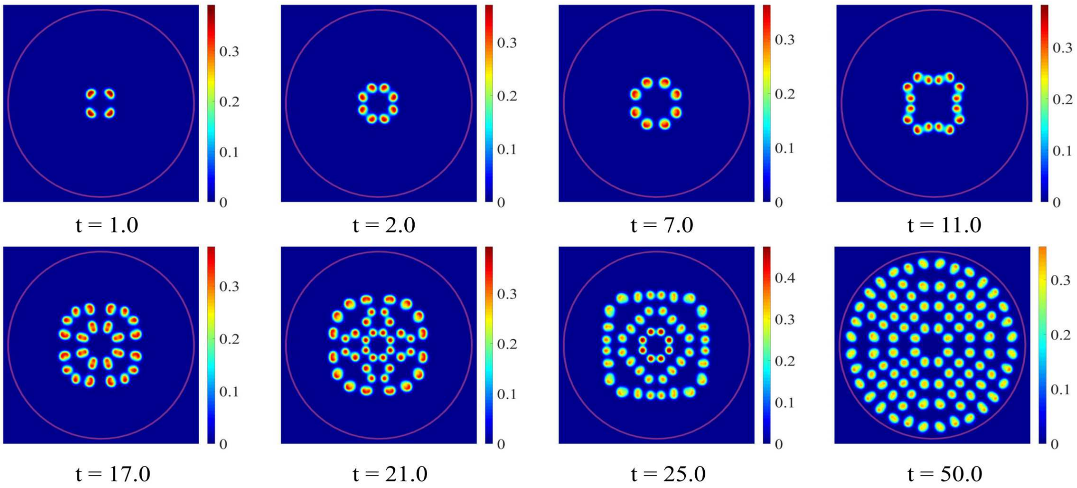


Figure 9: the radius $r = 1.8$, $T = 1,2,7,11,17,21,25,50$ on the 128×128 grid.The bounding box $\mathcal{B}$ is set to be $\mathcal{B} = (-2.0,2.0)\times(-2.0,2.0)$.

Example 4. This example solves the Dirichlet BVP of the Stokes equation on an sphere Ω which is given by

$$\Omega = \left\{ (x,y,z) \in \mathbb{R}^3 : \frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} < 1 \right\} \tag{5.2}$$

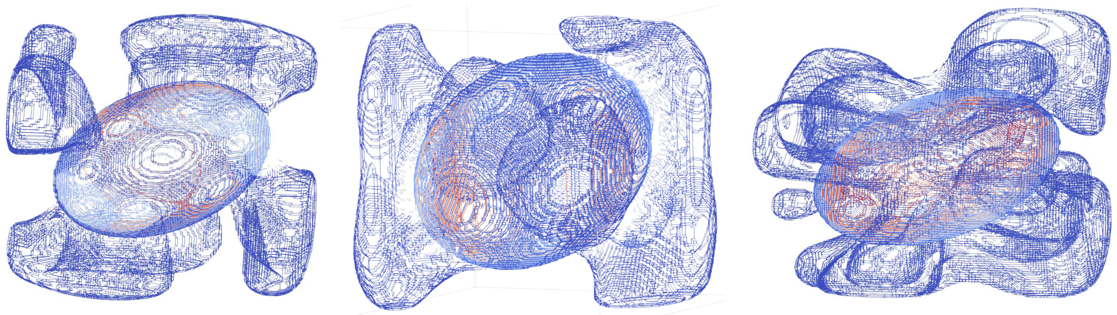
with $a = 1.0, b = 0.8, c = 0.6$. The bounding box $\mathcal{B}$ for the interface problem is $\mathcal{B} = [-1.2,1.2]\times[-1.2,1.2]\times[-1.2,1.2]$. The Dirichlet BC is chosen so that the exact solution reads

$$\begin{aligned} u^{(1)}(x,y,z) &= \exp(\cos y)+\exp(\sin z)-4\left(1-x^2-y^2\right)xy-4x^2z^2+\left(x^2+3z^2-2\right)\left(z^2-x^2\right)\\ u^{(2)}(x,y,z) &= \exp(\sin x)-4x^2y^2+\left(3x^2+y^2-2\right)\left(x^2-y^2\right)\\ u^{(3)}(x,y,z) &= \exp(\cos(x))-4\left(1-x^2-z^2\right)xz\\ p(x,y,z) &= \exp(\cos x+\sin y)+\exp(\cos z+\sin x)+8(3x^2-y^2)y+8x(3z^2-x^2) \end{aligned} \tag{5.3}$$

The error orders and execution times on the CPU and GPU are encapsulated in Table 5, derived from four distinct problem sizes: 32^3, 64^3, 128^3, 256^3, and 512^3. The table reveals an increasing GPU acceleration ratio with the escalation of computational scale. It is observed that enhanced performance is achieved for larger problems, indicating the scalability of the parallel method.

Boundary	grid size	32^3	64^3	128^3	256^3
Dirichlet	CPU time	44.85 s	272.39 s	1948.24 s	13521.36 s
	GPU time	1.34 s	2.82 s	4.98 s	38.62 s
	Speedup	33.47	96.59	391.21	350.11

Table 5: BVP of the Stokes equation on the bounding box $\mathcal{B} = (-1.2,1.2)\times(-1.2,1.2)\times(-1.2,1.2)$ with GMRES iterition method.



The velocity field $u^{(1)}$

The velocity field $u^{(2)}$

The velocity field $u^{(3)}$

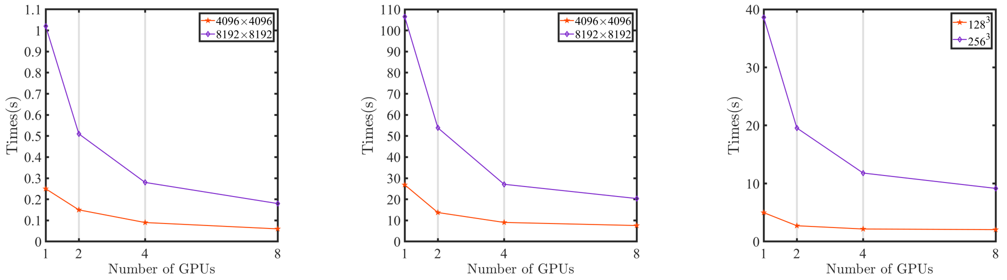
5.2 Multiple-GPU results

To augment computational precision, refinement is applied to the computational grid. Example 6 uses the same numerical test cases as examples 1, 3, and 4. Table 6 displays the computation times for solving the 2D Laplace, reaction-diffusion, and 3D Stokes equation. These computations were executed using 1, 2, 4, and 8 GPUs.

In the Fig. 10, we can conclude that multi-GPUs parallel computing achieves linear speedup, despite a slight decrease in single-GPU performance when more GPUs are employed, as shown in Tab. 6. The linear growth of parallel efficiency hindrance can be attributed to inter-GPU communication, involving tasks such as the distribution of boundary data(point 1 of procedure 2), exchange of ghost cell data(points 3 and 4 of procedure 2), and collection of boundary data(the point 6 of procedure 2) in 4.4.

equation	2D Laplace			2D reaction-diffusion			3D Stokes		
grid size	4096^2	8192^2	16384^2	4096^2	8192^2	16384^2	128^3	256^3	512^3
1GPU	0.25 s	1.02 s		26.87 s	106.51 s		4.98 s	38.62 s	
2GPUs	0.15 s	0.51 s		13.74 s	53.87 s		2.72 s	19.56 s	
4GPUs	0.09 s	0.28 s	1.01 s	9.05 s	27.12 s	123.78 s	2.16 s	11.8 s	
8GPUs	0.06 s	0.18 s	0.62 s	7.58 s	20.38 s	74.93 s	2.05 s	9.16 s	66.21 s

Table 6: Multi-GPUs execution time vs. single GPU.



(a) 2D Laplace

(b) 2D reaction-diffusion

(c) 3D Stokes

Figure 10: Comparison of parallel efficiency with different numbers of GPUs

6 Discussion

This paper presents a second-order, single, and multiple-GPU accelerated efficient KFBI method for elliptic boundary value problems. The equations are first transformed into a BIE, and then the unknown density in the equation is solved by GMRES iteration. Boundary and volume integral can be evaluated by equivalent interface problems to obtain the approximate solution. The procedure for solving the interface problem consists of four steps: discretization, correction, fast solving, and interpolation.

In the single GPU algorithm, since the KFBI method itself mainly focuses on the control points on the boundary and the irregular nodes near the boundary. We only need to assign threads to them and design a fast algorithm on the GPU to solve the interface problem efficiently. In the multiple-GPU algorithm, the system of linear equations in the interface problem must be solved by the ADM method, which involves the interaction of information between the host and the device handled by MPI.

The accuracy and efficiency of the algorithm are verified from numerical examples. The method is especially suited for GPU acceleration in large-scale computations, and the multiple-GPU distributed solver scales well. Numerical examples show that single-GPU solver speeds 50-200 times than traditional CPU while the eight GPUs distributed solver yields up to 60% parallel efficiency.

The single/multiple-GPU accelerated KFBI method can be extended for other PDEs, such as the Maxwell and elasticity equations. Furthermore, combined with the deep learning method, the KFBI method may exhibit potential applicability in solving equations within non-smooth domains on the GPU platform.

Acknowledgments

This work is financially supported by the Strategic Priority Research Program of Chinese Academy of Sciences(Grant No. XDA25010405). It is also partially supported by the National Key R&D Program of China, Project Number 2020YFA0712000, the National Natural Science Foundation of China (Grant No. DMS-11771290) and the Science Challenge Project of China (Grant No. TZ2016002). Additionally, it is supported by the Fundamental Research Funds for the Central Universities.

References

[1] H. Meuer, E. Strohmaier, J. Dongarra, H. Simon, and M. Meuer. Top500, November 2023.

[2] C. A. Navarro, N. Hitschfeld-Kahler, and L. Mateu. A survey on parallel computing and its applications in data-parallel problems using gpu architectures. *Communications in Computational Physics*, 15(2):285–329, 2014.

[3] W. Hwu. *GPU computing gems*. Waltham, MA : Morgan Kaufmann, Waltham, MA, jade ed.. edition, 2012.

[4] Y. Qian, C. Wang, and S. Zhou. A positive and energy stable numerical scheme for the poisson–nernst–planck–cahn–hilliard equations with steric interactions. *Journal of Computational Physics*, 426:109908, 2021.

[5] J. Ding, Z. Wang, and S. Zhou. Positivity preserving finite difference methods for poisson–nernst–planck equations with steric interactions: Application to slit-shaped nanopore conductance. *Journal of Computational Physics*, 397:108864, 2019.

[6] Y. Chai, K. Huang, S. Wang, Z. Xiang, and G. Zhang. The extrinsic enriched finite element method with appropriate enrichment functions for the helmholtz equation. *Mathematics*, 11:1664, 03 2023.

[7] L. Greengard and M. C. Kropinski. An integral equation approach to the incompressible navier-stokes equations in two dimensions. *SIAM Journal on Scientific Computing*, 20(1):318 – 336, 1998. Cited by: 37; All Open Access, Green Open Access.

[8] L. Quartapelle. Numerical solution of the incompressible navier-stokes equations. In *International series of numerical mathematics*, volume 113, 1993.

[9] S. Zhu, Q. Wu, and C. Liu. Shape and topology optimization for elliptic boundary value problems using a piecewise constant level set method. *Applied Numerical Mathematics*, 61(6):752–767, 2011.

[10] W. Gong, J. Li, and S. Zhu. Improved discrete boundary type shape gradients for pde-constrained shape optimization. *SIAM Journal on Scientific Computing*, 44(4):A2464–A2505, 2022.

[11] R. Chapko and R. Kress. Rothe's method for the heat equation and boundary integral equations. *Journal of Integral Equations and Applications*, 9(1):47 – 69, 1997. Cited by: 43; All Open Access, Bronze Open Access.

[12] Y. C. Zhou, S. Zhao, M. Feig, and G. W. We. High order matched interface and boundary method for elliptic equations with discontinuous coefficients and singular sources. *Journal of Computational Physics*, 213(1):1–30, 2006.

[13] H. Cheng, J. Huang, and T. J. Leiterman. An adaptive fast solver for the modified helmholtz equation in two dimensions. *Journal of Computational Physics*, 211(2):616–637, 2006.

[14] H. Sun and D. L. Darmofal. An adaptive simplex cut-cell method for high-order discontinuous galerkin discretizations of elliptic interface problems and conjugate heat transfer problems. *Journal of Computational Physics*, 278:445–468, 2014.

[15] P. García-Risueño, J. Alberdi-Rodriguez, M. J. T. Oliveira, X. Andrade, M. Pippig, J. Muguerza, A. Arruabarrena, and A. Rubio. A survey of the parallel performance and accuracy of poisson solvers for electronic structure calculations. *Journal of Computational Chemistry*, 35(6):427–444, 2014.

[16] W. Ying and C. S. Henriquez. A kernel-free boundary integral method for elliptic boundary value problems. *Journal of computational physics*, 227(2):1046–1074, 2007.

[17] W. Ying and W. Wang. A kernel-free boundary integral method for implicitly defined surfaces. *Journal of Computational Physics*, 252:606–624, 2013.

[18] W. Ying and W. Wang. A kernel-free boundary integral method for variable coefficients elliptic pdes. *Communications in Computational Physics*, 15(4):1108–1140, 2014.

[19] Y. Xie and W. Ying. A fourth-order kernel-free boundary integral method for the modified helmholtz equation. *Journal of Scientific Computing*, 78(3):1632–1658, 2019.

[20] Y. Xie, Z. Huang, and W. Ying. A cartesian grid based tailored finite point method for reaction-diffusion equation on complex domains. *Computers & Mathematics with Applications*, 97:298–313, 2021.

[21] Z. Zhao, H. Dong, and W. Ying. Kernel free boundary integral method for 3d incompressible flow and linear elasticity equations on irregular domains. *Computer Methods in Applied Mechanics and Engineering*, 414:116163, 2023.

[22] H. Dong, S. Li, W. Ying, and Z. Zhao. Kernel-free boundary integral method for two-phase stokes equations with discontinuous viscosity on staggered grids, 2023.

[23] H. Zhou, M. Huang, and W. Ying. Adi schemes for heat equations with irregular boundaries and interfaces in 3d with applications, 2023.

[24] R. Reddy and R. Banerjee. Gpu accelerated vof based multiphase flow solver and its application to sprays. *Computers & Fluids*, 117:287–303, 2015.

[25] S. K. Layton, A. Krishnan, and L. A. Barba. cuibm – A gpu-accelerated immersed boundary method. *CoRR*, abs/1109.3524, 2011.

[26] S. Liang, W. Liu, and L. Yuan. Solving seven-equation model for compressible two-phase flow using multiple gpus. *Computers & Fluids*, 99:156–171, 2014.

[27] C. Huang, B. Shi, N. He, and Z. Chai. Implementation of multi-gpu based lattice boltzmann method for flow through porous media. *Advances in Applied Mathematics and Mechanics*, 7(1):1–12, 2015.

[28] V. Volkov and B. Kazian. Fitting fft onto the g80 architecture. *Methodology*, January 2011.

[29] N. K. Govindaraju, B. Lloyd, Y. Dotsenko, B. Smith, and J. Manferdelli. High perfor-

mance discrete fourier transforms on graphics processors. In *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, pages 1–12, 2008.

[30] N. Nandapalan, J. Jaros, A. P. Rendell, and B. Treeby. Implementation of 3d ffts across multiple gpus in shared memory environments. In *2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 167–172, 2012.

[31] Y. Chen, X. Cui, and H. Mei. Large-scale fft on gpu clusters. In *Proceedings of the 24th ACM International Conference on Supercomputing*, ICS '10, page 315–324, New York, NY, USA, 2010. Association for Computing Machinery.

[32] A. Nukada, K. Sato, and S. Matsuoka. Scalable multi-gpu 3-d fft for tsubame 2.0 supercomputer. In *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–10, 2012.

[33] M. H. Aliabadi and P. H. Wen. *Boundary element methods in engineering and sciences*, volume 4. World Scientific, 2011.

[34] D. Yu. *Natural boundary integral method and its applications*, volume 539. Springer Science & Business Media, 2002.

[35] R. Kress, V. Maz'ya, and V. Kozlov. *Linear integral equations*, volume 82. Springer, 1989.

[36] G. C. Hsiao and W. L. Wendland. *Boundary integral equations*. Springer, 2008.

[37] G. Lin, F. Chen, P. Hu, X. Chen, J. Chen, J. Wang, and Z. Shi. Bi-greennet: Learning green's functions by boundary integral network, 2022.

[38] J. Wu and J. JaJa. High performance fft based poisson solver on a cpu-gpu heterogeneous platform. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pages 115–125, 2013.

[39] M. Naumov, L. S. Chien, P. Vandermersch, and U. Kapasi. Cusparse library. In *GPU Technology Conference*, 2010.

[40] N. K. Govindaraju, B. Lloyd, Y. Dotsenko, B. Smith, and J. Manferdelli. High performance discrete fourier transforms on graphics processors. In *SC'08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12. Ieee, 2008.

[41] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[42] P. A. Belov, E. R. Nugumanov, and S. L. Yakovlev. The arrowhead decomposition method for a block-tridiagonal system of linear equations. *Journal of Physics: Conference Series*, 929:012035, 11 2017.

[43] A. Gholami, D. Malhotra, H. Sundar, and G. Biros. Fft, fmm, or multigrid? a comparative study of state-of-the-art poisson solvers for uniform and nonuniform grids in the unit cube. *SIAM Journal on Scientific Computing*, 38(3):C280–C306, 2016.

[44] H. Dong, S. Li, W. Ying, and Z. Zhao. Second order convergence of a modified mac scheme for stokes interface problems, 2023.

[45] S. MacNamara and G. Strang. *Operator Splitting*, pages 95–114. Springer International Publishing, Cham, 2016.