



兰州大学

## 本科毕业论文（设计）

论文题目（中文） 利用格点 QCD 研究部分子分布函数  
——传播子的高性能求解

论文题目（英文） Studying parton distribution functions using lattice  
QCD——High-performance solution of propagators

学生姓名 张鑫

指导教师 于福升

学 院 核科学与技术学院

专 业 核科学与技术基地班（原子核物理方向）

年 级 2021 级

# 利用格点 QCD 研究部分子分布函数 ——传播子的高性能求解

## 中文摘要

**摘要：**格点量子色动力学（Lattice Quantum Chromodynamics，简称格点 QCD）作为目前较为成熟的研究量子色动力学非微扰问题的基准性数值方法，在强相互作用物理研究中具有不可忽略的作用。本课题聚焦于该方法的核心应用场景——部分子分布函数的数值研究，并针对其中的关键计算瓶颈即传播子的高效求解展开攻关。从数学形式上看，该问题可归结为具有特殊结构的大规模稀疏复系数矩阵线性方程组的求解，其维度规模在传统通用计算框架下已超出可处理范围，亟需结合其数学特性设计专用的高性能并行求解算法，这正是本课题研究的核心目标与技术突破点。目前研究工作已取得阶段性成果：首先完成了格点 QCD 中两类典型作用量对应的稀疏矩阵构建器开发——基于 Wilson 作用量的 Dslash 算符（格点 QCD 中描述夸克场传播的特殊微分算符）及其经 Clover 作用量优化的改进型版本；其次针对上述矩阵特性，成功实现了共轭梯度法（Conjugate Gradient Method, CG）与稳定双共轭梯度法（Biconjugate Gradient Stabilized Method, BICGSTAB）两类核心线性求解算法；最后构建了面向实际应用的 Python 计算接口（访问地址：<https://gitee.com/zhangxin8069/PyQCU>），为后续大规模物理问题求解奠定了完整的软件基础。

**关键词：**理论物理，第一性原理计算，LQCD，高性能算法，CUDA 并行软件

# Studying parton distribution functions using lattice QCD

## ——High-performance solution of propagators

### Abstract

Lattice Quantum Chromodynamics (Lattice QCD for short), as a relatively mature benchmark numerical method for studying non-perturbation problems in quantum chromodynamics at present, plays an indispensable role in the research of strong interaction physics. This project focuses on the core application scenario of this method - the numerical study of some molecular distribution functions, and conducts research and development on the key computational bottleneck among them, namely the efficient solution of propagators. From a mathematical perspective, this problem can be reduced to the solution of linear equations of large-scale sparse complex coefficient matrices with special structures. The dimensional scale of this problem has exceeded the manageable range under the traditional general computing framework. It is urgent to design a dedicated high-performance parallel solution algorithm in combination with its mathematical characteristics. This is precisely the core goal and technical breakthrough point of this research topic. The current research work has achieved phased results: Firstly, the development of sparse matrix builders corresponding to two types of typical actions in lattice QCD has been completed - the Dslash operator based on Wilson action (a special differential operator describing the propagation of quark fields in lattice QCD) and its improved version optimized by Clover action. Secondly, in view of the above-mentioned matrix characteristics, Two types of core linear solution algorithms, namely the Conjugate Gradient Method (CG) and the Biconjugate Gradient Stabilized Method (BICGSTAB), have been successfully implemented. Finally built a Python calculation for actual application interface (access address: <https://gitee.com/zhangxin8069/PyQCU>), for the subsequent mass physical problem solving the complete software foundation.

**Keywords:** Theoretical physics, First-principles Computation, LQCD, High-performance algorithm, CUDA parallel software.

# 目 录

引 言.....	4
第一章 研究背景.....	5
第二章 理论框架.....	5
2.1 格点量子色动力学.....	5
2.2 格点中的部分子分布函数.....	6
2.3 格点中的传播子.....	8
2.4 格点中的 Dslash .....	9
2.4.1 Wilson Dslash .....	9
2.4.2 Clover Dslash .....	9
2.5 求解算法.....	10
2.5.1 CG 算法.....	11
2.5.2 BISTABCG 算法.....	12
第三章 高性能实现.....	12
3.1 实现思路.....	12
3.2 Wilson Dslash 的实现 .....	13
3.3 Clover Dslash 的实现 .....	14
3.4 求解算法的实现.....	15
3.4.1 CG 算法的实现.....	15
3.4.2 BISTABCG 算法的实现.....	15
3.5 性能优化.....	16
3.5.1 优化 1.....	16
3.5.2 优化 2.....	17
3.5.3 优化 3.....	17
3.5.4 优化 4.....	18
第四章 结果产生与分析.....	19

4.1 产生与分析思路.....	19
4.2 Wilson Dslash 的结果 .....	20
4.3 Clover Dslash 的结果 .....	22
4.4 求解算符的结果.....	23
4.4.1 CG 算法的结果.....	23
4.4.2 BISTABCG 算法的结果.....	24
4.5 总体分析.....	25
<b>第五章 用户配置.....</b>	<b>26</b>
5.1 PyQCU.....	26
5.2 Docker.....	27
<b>第六章 结论与展望.....</b>	<b>27</b>
<b>参考文献.....</b>	<b>28</b>
<b>致 谢.....</b>	<b>28</b>

## 引 言

长期以来,格点QCD作为研究部分子分布函数(PDFs)的有力工具得到了越来越多的重视。在国内,格点QCD方兴未艾,相关的物理工作成果频出,但是作为格点QCD基础工程的高性能软件,却相对来说其国产化进度较为落后。本课题正是聚焦于利用格点QCD研究部分子分布函数的关键计算瓶颈,即传播子的高效求解展开攻关,以期使用完全自研自产的国产代码实现国外先进软件的基础功能。

## 第一章 研究背景

格点QCD将物理的时空离散化成有限数目的格点。以常用的 $32*32*32*64$ 格点为例，当研究部分子分布函数时，我们需要在波函数上作用算符，体现在实际工作中我们需要对上述格点向量乘一个矩阵。泛泛来说，上述向量的大小为 $402.653184\text{MB}$ （此向量带有自旋维度与色维度且为复数，以双精度存储，即 $32*32*32*64*4*3*2*8\text{B}=402.653184\text{MB}$ ），而上述矩阵的大小为 $10.133099161583616\text{PB}$ （此矩阵同样带有自旋维度与色维度且为复数，以双精度存储，即 $32*32*32*64*4*3*32*32*32*64*4*3*2*8\text{B}=10.133099161583616\text{PB}$ ）。可见此向量大小存储方能接受，但此矩阵存储的成本已经是天文数字，这二者间矩阵乘的计算成本更是在传统通用计算框架下已超出可处理范围。然而上述的矩阵乘在利用格点QCD研究部分子分布函数的计算传播子步骤中要进行200-300次，这也仅仅是一次物理工作中非常基础的步骤，可见高性能软件对于上述工作的重要性。

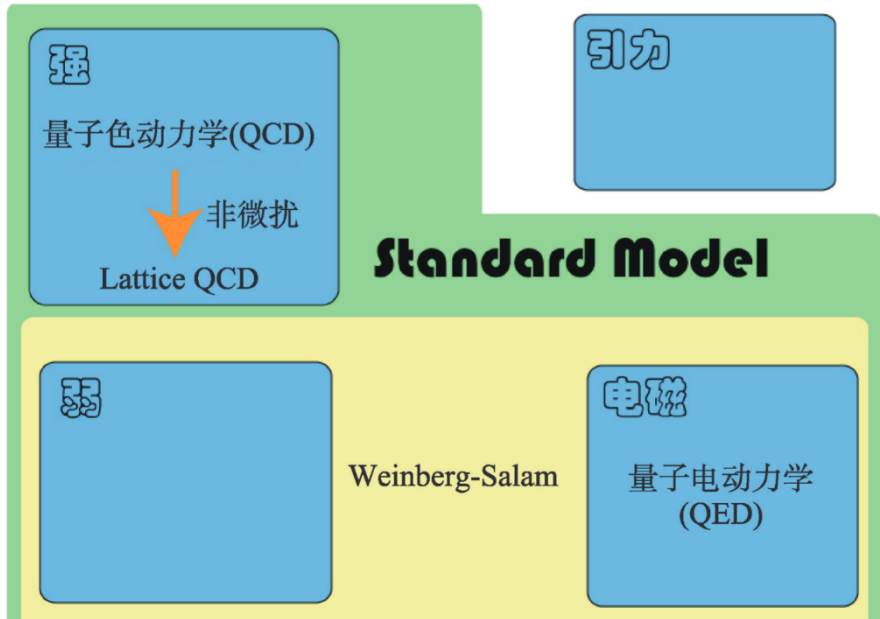
## 第二章 理论框架

### 2.1 格点量子色动力学

在QCD中拉氏量为：

$$\mathcal{L}_{QCD} = \overline{\psi}_i \left( i\gamma^\mu (D_\mu)_{ij} - m\delta_{ij} \right) \psi_j - \frac{1}{4} F_{\mu\nu}^a F_a^{\mu\nu} \quad (1)$$

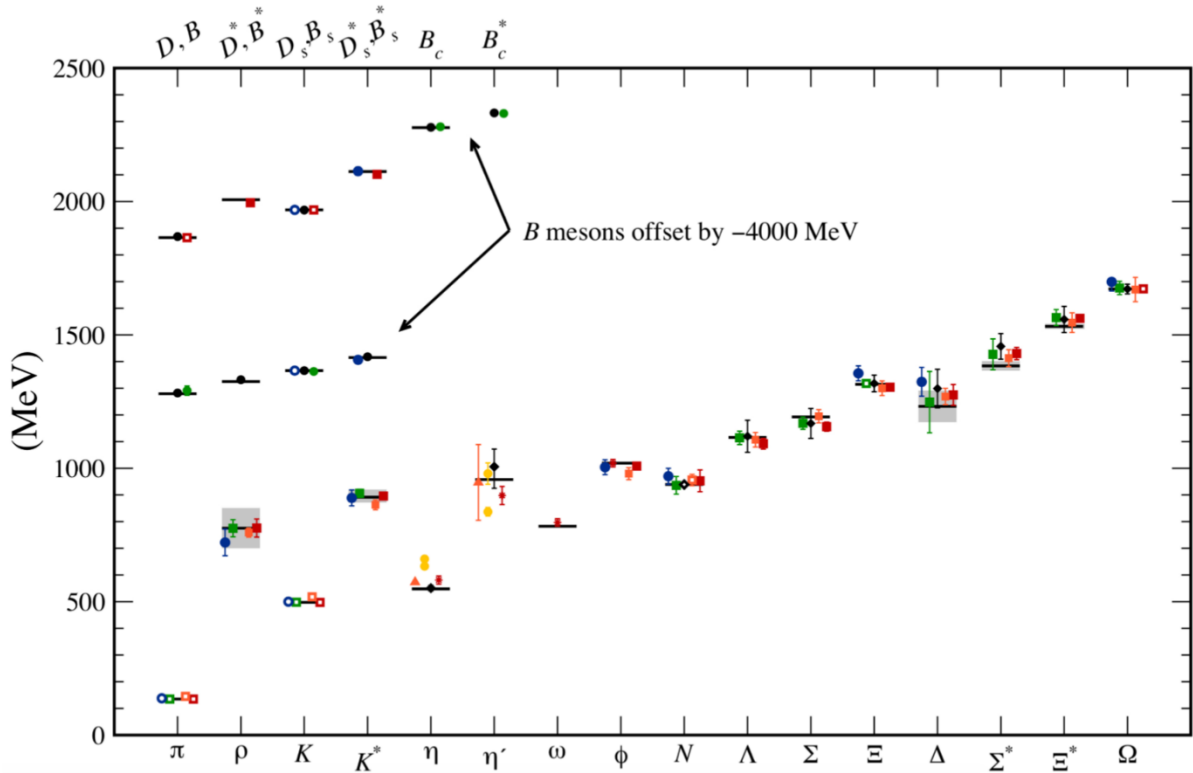
其中 $\psi_i$ 为夸克场， $\gamma^\mu$ 为狄拉克矩阵， $D_\mu$ 为协变导数， $F_{\mu\nu}^a$ 为规范胶子场张量。



图表 1 格点量子色动力学在标准模型中的相对位置的示意图[1]

其中QCD是一种强相互作用的规范场论，而LQCD的计算方法就是在不破坏规范不变性的情况下，将连续的时空离散化为欧几里得时空下的四维超立方格子包含特定的信息，包括其位置，自旋和颜色等。将其位置矢量定义为 $x_\mu = n_\mu a$ ，其中 $n_\mu$ 是两格子之间相邻的格子数， $a$ 被称为格距[1]。

格点的计算方法在非微扰QCD中得到了广泛应用，但其适用范围不仅仅局限于非微扰QCD。其核心思想是将连续时空离散化为分立时空，这才是格点方法最重要的部分。此外，它具有天然的优势：在构建量子场论时，由于非平凡的量子场存在紫外发散的困难，导致量子场论在构建过程中缺乏严格的数学定义，因此必须要有一个内禀的紫外截断来解决紫外发散的问题，而在LQCD中因为格距 $a$ 的存在所以天然的存在一个紫外截断 $\Lambda = \pi/a$ ，而又因为整个空间的体积被限制在一个特定大小的四维格子内部，所以也就存在一个红外的截断 $\Lambda' = 2\pi/L$ ， $L$ 为空间的长度[1]。



图表 2 格点 QCD 计算得到的强子谱[2]

现在，通过格点 QCD 进行的强子谱计算已经在国际上被广泛认可，如上图，可以看到各大格点合作组确定的强子谱基本都与实验相符[2]。

## 2.2 格点中的部分子分布函数

根据费曼的部分子模型，接近光速运动的核子可以看作是由高速运动的自由点粒子构成，但是在这些高速运动的粒子中，夸克所携带的质子动量的份额 $x$ 是不确定的，这是因为

夸克之间会有相互作用导致动量发生变化。但夸克的动量具有一定的概率分布，因此为了定量描述夸克的动量分布，引入部分子分布函数 (PDFs)  $\delta f(x, \mu)$ ， $x$  为夸克携带的动量分数， $\delta f(x, \mu)dx$  表示为动量在  $x \rightarrow x + dx$  之间的夸克数目。

但是当夸克的动量非常大时，其会遇到紫外发散的困难，所以可以在天然具有紫外截断的 LQCD 中计算有限大动量的准 PDFs (quasi PDFs)，然后利用大动量外推计算无穷大动量时的 PDFs[3]。(这里仅考虑价夸克的部分子分布函数，而不考虑胶子的部分子分布函数)

准 PDFs 被定义为：

$$\delta \tilde{f}(x, P_z, 1/a) = \int \frac{dz}{4\pi} e^{ixzP_z} \langle P | \bar{\psi}(z) \gamma^t U(z, 0) \psi(0) | P \rangle \quad (2)$$

$P_z$  是沿着  $z$  方向的动量， $P$  为夸克携带的动量， $\psi(0)$  表示为位置在 0 点的夸克场， $U(z, 0)$  为位置 0 点与  $z$  点的规范链接。

以  $\pi^+$  介子为例，为了提取 PDFs 的矩阵元，将其两点 and 三点关联函数进行分解可以得到：

$$\begin{aligned} C_2^{\pi^+}(P_z, t, t_{sep}) &= |\mathcal{A}_0|^2 \frac{e^{-E_\pi \vec{P} \frac{L_t t}{2}}}{E_\pi \vec{P}} \cosh \left[ -E_\pi \vec{P} \left( t - \frac{L_t}{2} \right) \right] + \dots \\ C_3^{\pi^+}(P_z, t, t_{sep}) &= |\mathcal{A}_0|^2 \frac{e^{-E_\pi \vec{P} \frac{L_t t}{2}}}{E_\pi \vec{P}} \langle 0 | \mathcal{O}_\Gamma | 0 \rangle \cosh \left[ -E_\pi \vec{P} \left( t - \frac{L_t}{2} \right) \right] + \dots \quad (3) \end{aligned}$$

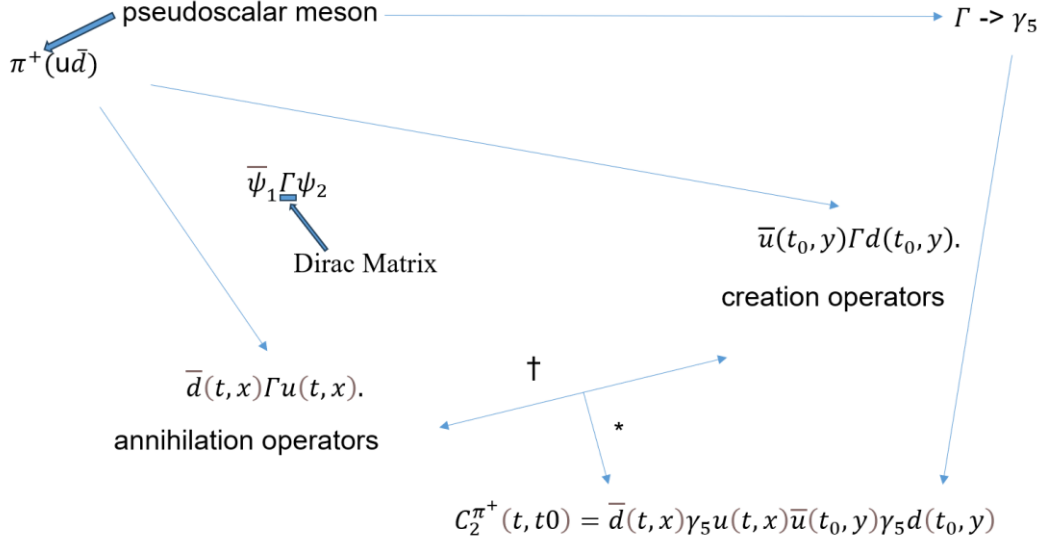
其中  $\langle 0 | \mathcal{O}_\Gamma | 0 \rangle$  是准 PDFs 的矩阵元， $t_{sep}$  表示源和汇的时间间隔， $t$  表示两点关联函数所处的时间片和算符  $\mathcal{O}_\Gamma$  插入时的时间， $E_0$  表示基态的能量，省略号表示在强子中衰减速度相较于基态快很多的高激发态的贡献[4]。



## 2.3 格点中的传播子

继续以 $\pi^+$ 介子的两点关联函数为例，其还可以表示为：

$$C_2^{\pi^+}(t, t_0) = \bar{d}(t, x) \gamma_5 u(t, x) \bar{u}(t_0, y) \gamma_5 d(t_0, y) \quad (4)$$

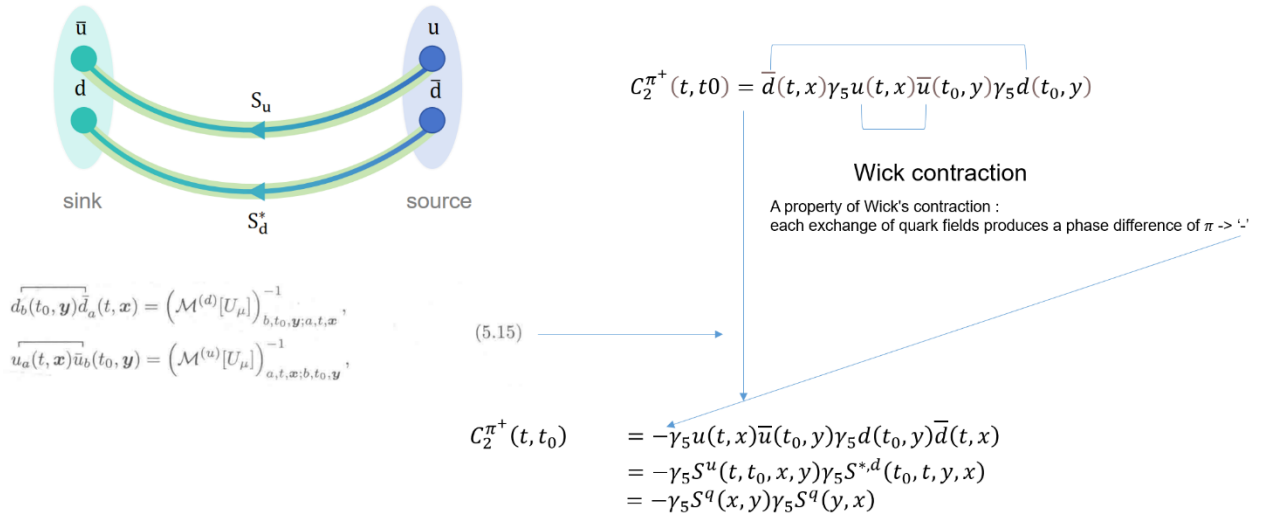


图表 3  $\pi^+$ 介子的两点关联函数的推导示意图

对其进行维克收缩后表示为：

$$C_2^{\pi^+}(t, t_0) = -\gamma_5 S^q(x, y) \gamma_5 S^q(y, x) \quad (5)$$

其中 $S$ 被称为传播子，其具体表现为费米子矩阵的逆 $\mathcal{M}^{-1}$ [1]。



图表 4 对 $\pi^+$ 介子的两点关联函数维克收缩示意图

结合（3）式与（5）式，我们可以通过求解传播子 $S$ 来给出两点关联函数 $C_2^{\pi^+}$ ，再结合

其与准 PDFs 的矩阵元的关系来研究部分子分布函数。

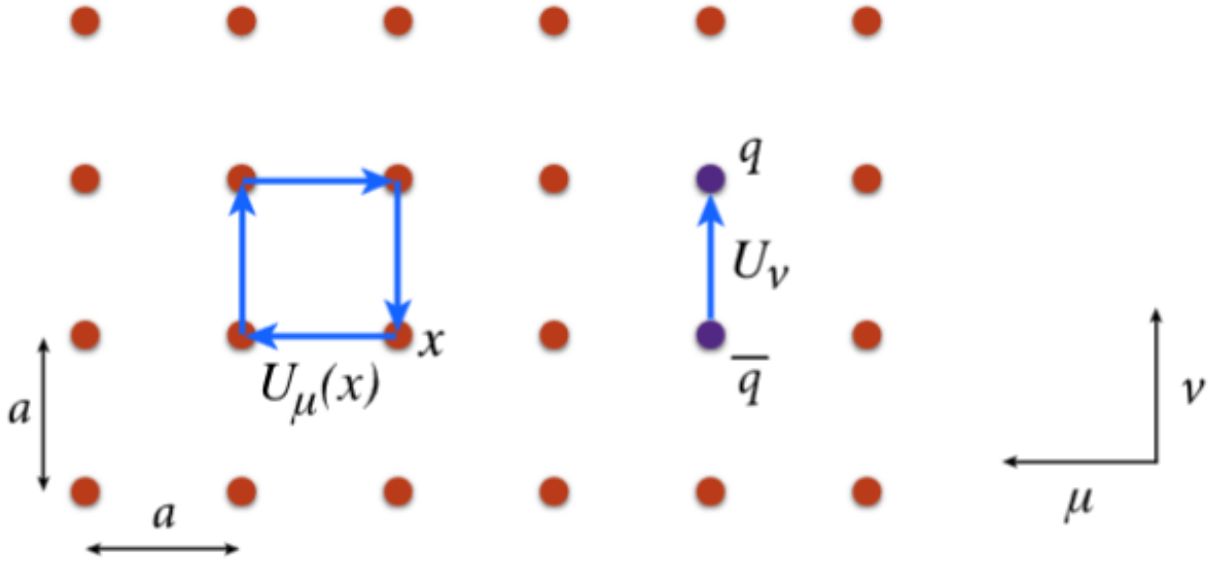
## 2.4 格点中的 Dslash

上述的费米子矩阵  $\mathcal{M}$  或者说其与向量进行矩阵乘的操作，我们一般称为 Dslash。依据选用的格点上的作用量的不同，我们常用的 Dslash 有 Wilson Dslash 与 Clover Dslash。

### 2.4.1 Wilson Dslash

当使用 Wilson 作用量时，给出 Wilson Dslash[2]：

$$M_{x,y}^W[U]a = \delta_{xy} - \kappa \sum_{\mu} [(r - \gamma_{\mu}) U_{x,\mu} \delta_{x,y-\mu} + (r + \gamma_{\mu}) U_{x-\mu,\mu}^{\dagger} \delta_{x,y+\mu}] \quad (6)$$



图表 5 二维格点上的物质场和规范链接[2]

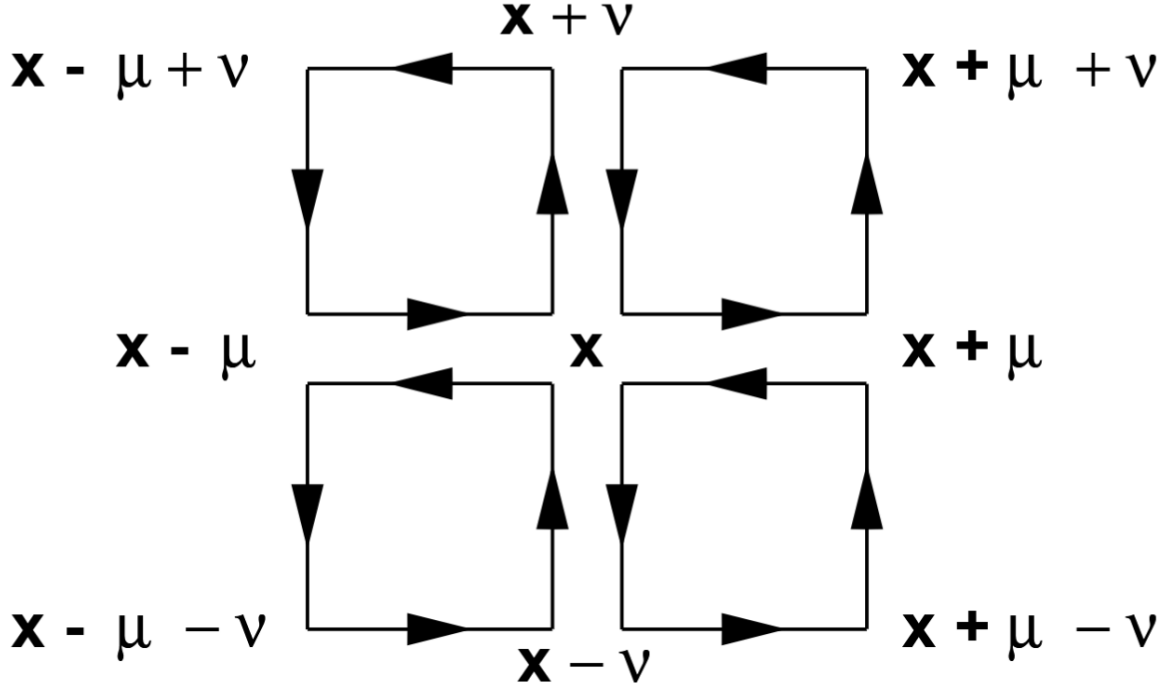
其中  $\kappa$  与  $a$  为实际使用时给出的系数，在此不做讨论， $U_{\mu}$  为格点上的规范链接，后续将作为输入之一。

### 2.4.2 Clover Dslash

当使用 Clover 作用量时，给出 Clover Dslash[2]：

$$M_{x,y}^W[U]a =$$

$$\delta_{xy} - \kappa \sum_{\mu} [(r - \gamma_{\mu}) U_{x,\mu} \delta_{x,y-\mu} + (r + \gamma_{\mu}) U_{x-\mu,\mu}^{\dagger} \delta_{x,y+\mu}] - \frac{iaC_{SW}kr}{4} \sigma_{\mu\nu} F_{\mu\nu} \delta_{x,y} \quad (7)$$



图表 6 Clover(四叶草)项示意图[2]

其中,  $F_{\mu\nu} = \frac{1}{4} \sum_p \frac{1}{2} [U_p(x) - U_p^\dagger(x)]$ ,  $U_p$  展开为:

$$\begin{aligned} U_1 &= u(x, \mu)u(x + \mu, \nu)u^\dagger(x + \nu, \mu)u^\dagger(x, \nu) \\ U_2 &= u(x, \nu)u^\dagger(x - \mu + \nu, \mu)u^\dagger(x - \mu, \nu)u(x - \mu, \mu) \\ U_3 &= u^\dagger(x - \mu, \mu)u^\dagger(x - \mu - \nu, \nu)u(x - \mu - \nu, \mu)u(x - \nu, \nu) \\ U_4 &= u^\dagger(x - \nu, \nu)u(x - \nu, \mu)u(x - \nu + \mu, \nu)u^\dagger(x, \mu) \end{aligned}$$

且  $\sigma_{\mu\nu} = \gamma_\mu \gamma_\nu - \gamma_\nu \gamma_\mu = 2\gamma_\mu \gamma_\nu$ , 则,  $\sigma_{\mu\nu} F_{\mu\nu}$  可展开为:

$$\begin{aligned} \sigma_{\mu\nu} F_{\mu\nu} &= 2\gamma_\mu \gamma_\nu \frac{1}{4} \sum_p \frac{1}{2} [U_p(x) - U_p^\dagger(x)] \\ &= \frac{1}{4} \gamma_\mu \gamma_\nu [u(x, \mu)u(x + \mu, \nu)u^\dagger(x + \nu, \mu)u^\dagger(x, \nu) \\ &\quad + u(x, \nu)u^\dagger(x - \mu + \nu, \mu)u^\dagger(x - \mu, \nu)u(x - \mu, \mu) \\ &\quad + u^\dagger(x - \mu, \mu)u^\dagger(x - \mu - \nu, \nu)u(x - \mu - \nu, \mu)u(x - \nu, \nu) \\ &\quad + u^\dagger(x - \nu, \nu)u(x - \nu, \mu)u(x - \nu + \mu, \nu)u^\dagger(x, \mu) - \text{BEFORE}^\dagger] \quad (8) \end{aligned}$$

## 2.5 求解算法

综合 2.3 与 2.4 所述, 传播子  $S$  为费米子矩阵的逆  $\mathcal{M}^{-1}$ , 即 Dslash 的逆。在实际应用中我们会给定一个向量作为输入, 即求解传播子的问题可以简化为求解  $Ax = b$ 。求解此类问题有 CG 算法与 BISTABCG 算法两种较为成熟的简单方法 (实际运用时使用进阶的 MultiGrid 算法)。

### 2.5.1 CG 算法

CG算法一般作为最为基础的求解器，常用作进阶算法（例如MultiGrid算法）的内嵌求解器，但其要求 $A$ 为正定矩阵，我们常常通过求解 $A^\dagger Ax = A^\dagger b$ 来满足这个要求。

---

#### 算法 1 Conjugate gradient

---

<pre> 1: <b>procedure</b> CG(<math>A, b</math>) 2:   <math>x \leftarrow 0</math> 3:   <math>r \leftarrow b - Ax</math> 4:   <b>If</b> converge <b>return</b> <math>x</math> 5:   <math>p \leftarrow r</math> 6:   <b>while</b> not reach max iteration <b>do</b> 7:     <math>\alpha \leftarrow \frac{\langle r, r \rangle}{\langle p, Ap \rangle}</math> 8:     <math>x \leftarrow x + \alpha p</math> 9:     <math>r' \leftarrow r - \alpha Ap</math> 10:    <b>If</b> converge <b>return</b> <math>x</math> 11:    <math>\beta \leftarrow \frac{\langle r', r' \rangle}{\langle r, r \rangle}</math> 12:    <math>p \leftarrow r' + \beta p</math> 13:    <math>r \leftarrow r'</math> 14:  <b>end while</b> 15: <b>end procedure</b> </pre>	<p><math>\triangleright</math> Solve <math>Ax = b</math> by CG</p>
---	--

---

图表 7 CG 算法的伪代码示意图[2]

### 2.5.2 BISTABCG 算法

BISTABCG算法不要求A为正定矩阵,也可以作为进阶算法的内嵌求解器,且相较于CG算法更加稳定可靠。当实现进阶算法复杂度过大时,通过实现BISTABCG算法来解决问题具有相当的“性价比”。

---

#### 算法 3 Biconjugate gradient stablized

---

<pre> 1: <b>procedure</b> BiCGSTAB(<math>A, b</math>) 2:   <math>x \leftarrow 0</math> 3:   <math>r \leftarrow r_0 \leftarrow b - Ax</math> 4:   <b>If</b> converge <b>return</b> <math>x</math> 5:   <math>p \leftarrow r</math> 6:   <b>while</b> not reach max iteration <b>do</b> 7:     <math>\alpha \leftarrow \frac{\langle r_0, r \rangle}{\langle r_0, Ap \rangle}</math> 8:     <math>x \leftarrow x + \alpha p</math> 9:     <math>r' \leftarrow r - \alpha Ap</math> 10:    <b>If</b> converge <b>return</b> <math>x</math> 11:    <math>t \leftarrow Ar</math> 12:    <math>\omega \leftarrow \frac{\langle t, r \rangle}{\langle t, t \rangle}</math> 13:    <math>x \leftarrow x + \omega r'</math> 14:    <math>r' \leftarrow r' - \omega Ar'</math> 15:    <b>If</b> converge <b>return</b> <math>x</math> 16:    <math>\beta \leftarrow \frac{\langle r', r' \rangle}{\langle r, r \rangle}</math> 17:    <math>p \leftarrow r' + (\alpha\beta/\omega)p - \alpha\beta Ap</math> 18:    <math>r \leftarrow r'</math> 19:  <b>end while</b> 20: <b>end procedure</b> </pre>	<p>▷ Solve <math>Ax = b</math> by BiCGSTAB</p>
---	--

---

图表 8 BISTABCG 算法的伪代码示意图[2]

## 第三章 高性能实现

### 3.1 实现思路

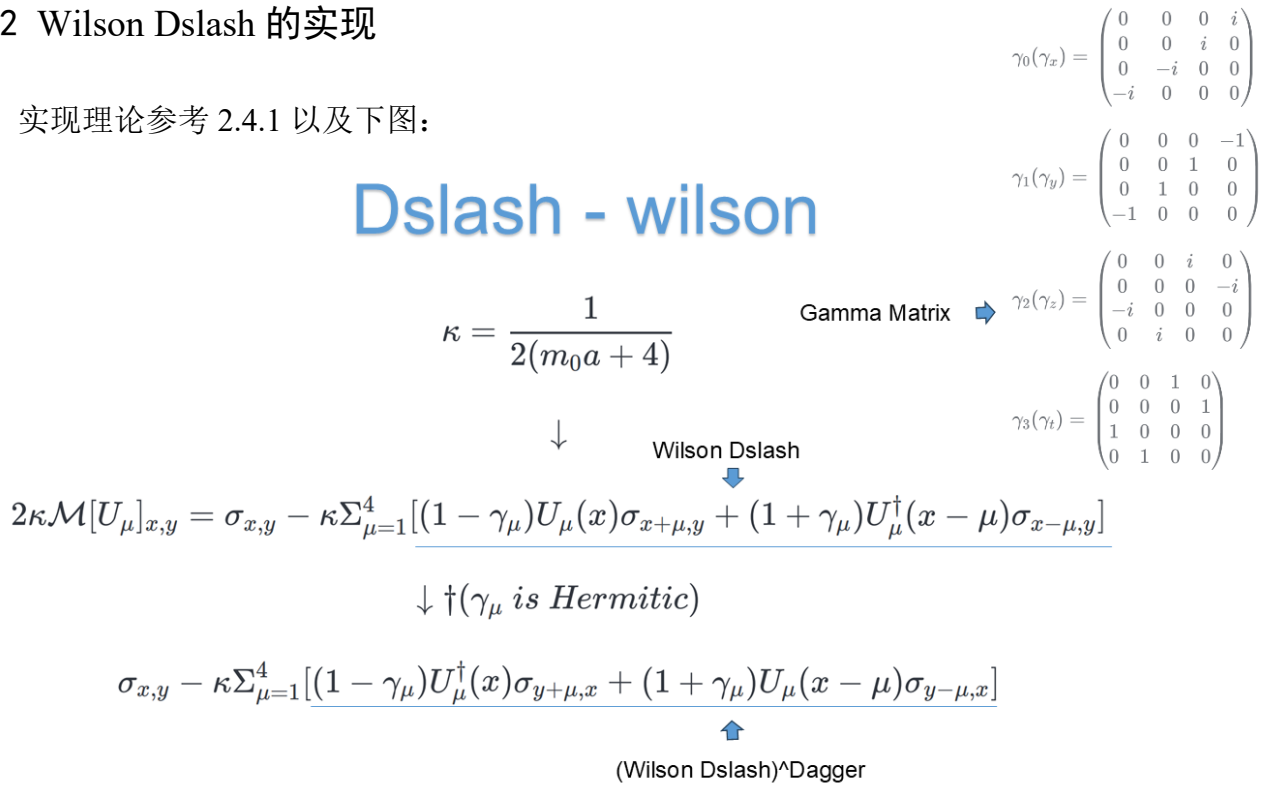
结合当下软件生态与国内硬件环境,我们计划编写使用MPI通讯的多GPU多流的CUDA C++程序。

分析如下:

1. 美国禁运nvlink（一种软硬件结合的高性能通讯方案）的配套设备，只能使用开源的MPI协议。
2. 单个GPU对于当前的格点QCD工作来说杯水车薪（格子过大），多GPU势在必行。
3. CUDA生态在高性能计算领域处于绝对垄断地位无法绕开，但第三方（例如国内的曙光DCU）相似接口的生态正在逐步形成。
4. 多流为CUDA程序的一个优化方案，可以提高一定程度上的性能。
5. CUDA生态有多个接口，其中C++接口虽然较为复杂但是优化潜力最大。

### 3.2 Wilson Dslash 的实现

实现理论参考 2.4.1 以及下图：



图表 9 Wilson Dslash 实现示意图

核心代码参考 wilson\_dslash.cu，这里简单起见，只给出接口（后续接口类似，只给出此一例）：

1. `#include "../include/qcu.h"`
2. `#pragma optimize(5)`
3. `using namespace qcu;`
4. `using T = float;`
5. `void applyDslashQcu(void * fermion_out, void * fermion_in, void * gauge, QcuParam * param, int parity, QcuParam * grid)`
6. `{`
7. `// define for apply_wilson_dslash`

```

8.  LatticeSet < T > _set;
9.  _set.give(param->lattice_size,grid->lattice_size,parity);
10. _set.init();
11. dptzyxcc2ccdptzyx < T > (gauge,&_set);
12. tzyxsc2sctzyx < T > (fermion_in,&_set);
13. tzyxsc2sctzyx < T > (fermion_out,&_set);
14. LatticeWilsonDslash < T > _wilson_dslash;
15. _wilson_dslash.give(&_set);
16. _wilson_dslash.run_test(fermion_out,fermion_in,gauge);
17. ccdptzyx2dptzyxcc < T > (gauge,&_set);
18. sctzyx2tzyxsc < T > (fermion_in,&_set);
19. sctzyx2tzyxsc < T > (fermion_out,&_set);
20. _set.end();
21. }

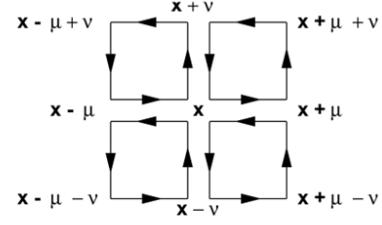
```

核心代码托管在 ‘[gitee.com/zhangxin8069/qcu/blob/dev26/src](https://gitee.com/zhangxin8069/qcu/blob/dev26/src)’ (下同), 共计1400行, 性能测试参考4.2。

### 3.3 Clover Dslash 的实现

实现理论参考 2.4.2 以及下图:

Dslash - clover



$$\begin{aligned}
 \sigma_{\mu\nu} F_{\mu\nu} &= 2\gamma_\mu \gamma_\nu \frac{1}{4} \sum_p \frac{1}{2} [U_p(x) - U_p^\dagger(x)] \\
 &= \frac{1}{4} \gamma_\mu \gamma_\nu [ \\
 &\quad u(x, \mu) u(x + \mu, \nu) u^\dagger(x + \nu, \mu) u^\dagger(x, \nu) \\
 &\quad + u(x, \nu) u^\dagger(x - \mu + \nu, \mu) u^\dagger(x - \mu, \nu) u(x - \mu, \mu) \\
 &\quad + u^\dagger(x - \mu, \mu) u^\dagger(x - \mu - \nu, \nu) u(x - \mu - \nu, \mu) u(x - \nu, \nu) \\
 &\quad + u^\dagger(x - \nu, \nu) u(x - \nu, \mu) u(x - \nu + \mu, \nu) u^\dagger(x, \mu) - BEFORE^\dagger ]
 \end{aligned}$$

$U_1 = u(x, \mu) u(x + \mu, \nu) u^\dagger(x + \nu, \mu) u^\dagger(x, \nu)$   
 $U_2 = u(x, \nu) u^\dagger(x - \mu + \nu, \mu) u^\dagger(x - \mu, \nu) u(x - \mu, \mu)$   
 $U_3 = u^\dagger(x - \mu, \mu) u^\dagger(x - \mu - \nu, \nu) u(x - \mu - \nu, \mu) u(x - \nu, \nu)$   
 $U_4 = u^\dagger(x - \nu, \nu) u(x - \nu, \mu) u(x - \nu + \mu, \nu) u^\dagger(x, \mu)$

Before the '['  $\downarrow$

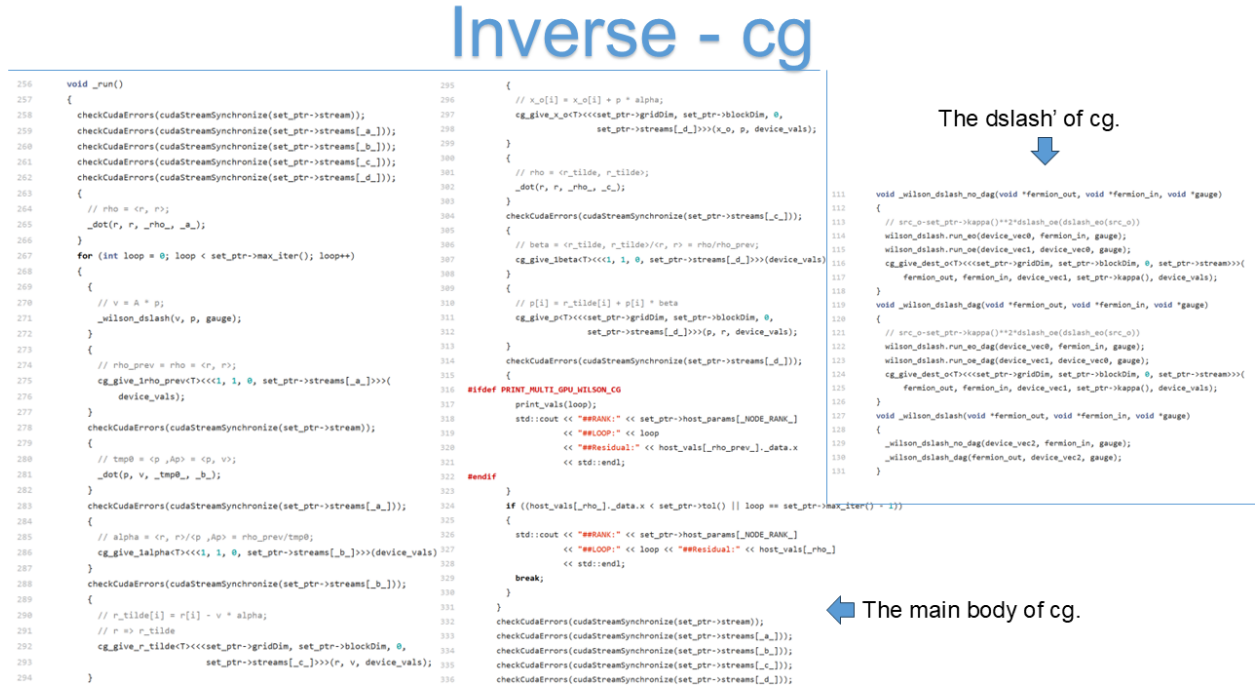
图表 10 Clover Dslash 实现示意图

核心代码参考 `clover_dslash_single.cu`、`clover_dslash_comm.cu`、`clover_dslash_multi.cu` 共计 3643 行, 性能测试参考 4.3。

## 3.4 求解算法的实现

### 3.4.1 CG 算法的实现

实现理论参考 2.5.1 以及下图：



图表 11 CG 算法实现示意图

核心代码参考 lattice\_cg.h, 托管在 ‘[gitee.com/zhangxin8069/qcu/blob/dev26/include](https://gitee.com/zhangxin8069/qcu/blob/dev26/include)’ (下同), 共计 420 行, 性能测试参考 4.4.1。

### 3.4.2 BISTABCG 算法的实现

实现理论参考 2.5.2 以及下图：



# Inverse - bicgstab



图表 12 BISTABCG 算法实现示意图

核心代码参考 lattice\_bicstab.h, 共计 415 行, 性能测试参考 4.4.2。

## 3.5 性能优化

### 3.5.1 优化 1

在进行 Dslash 操作时, 其自旋维度体现为  $\gamma$  矩阵, 其形式如下图左半部分所示:

$$1 + \gamma_0(\gamma_x) = \begin{pmatrix} 1 & 0 & 0 & i \\ 0 & 1 & i & 0 \\ 0 & -i & 1 & 0 \\ -i & 0 & 0 & 1 \end{pmatrix} \rightarrow X-1$$

$$1 + \gamma_1(\gamma_y) = \begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix} \rightarrow Y-1$$

$$1 + \gamma_2(\gamma_z) = \begin{pmatrix} 1 & 0 & i & 0 \\ 0 & 1 & 0 & -i \\ -i & 0 & 1 & 0 \\ 0 & i & 0 & 1 \end{pmatrix} \rightarrow Z-1$$

$$1 + \gamma_3(\gamma_t) = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \rightarrow t-1$$

$$1 - \gamma_0(\gamma_x) = \begin{pmatrix} 1 & 0 & 0 & -i \\ 0 & 1 & -i & 0 \\ 0 & i & 1 & 0 \\ i & 0 & 0 & 1 \end{pmatrix} \rightarrow X+1$$

$$1 - \gamma_1(\gamma_y) = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \rightarrow Y+1$$

$$1 - \gamma_2(\gamma_z) = \begin{pmatrix} 1 & 0 & -i & 0 \\ 0 & 1 & 0 & i \\ i & 0 & 1 & 0 \\ 0 & -i & 0 & 1 \end{pmatrix} \rightarrow Z+1$$

$$1 - \gamma_3(\gamma_t) = \begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix} \rightarrow t+1$$

Trick - gamma  
for  
Dslash - wilson

$$1 - \gamma_0(\gamma_x) = \begin{pmatrix} 1 & 0 & 0 & -i \\ 0 & 1 & -i & 0 \\ 0 & i & 1 & 0 \\ i & 0 & 0 & 1 \end{pmatrix} \rightarrow X+1$$

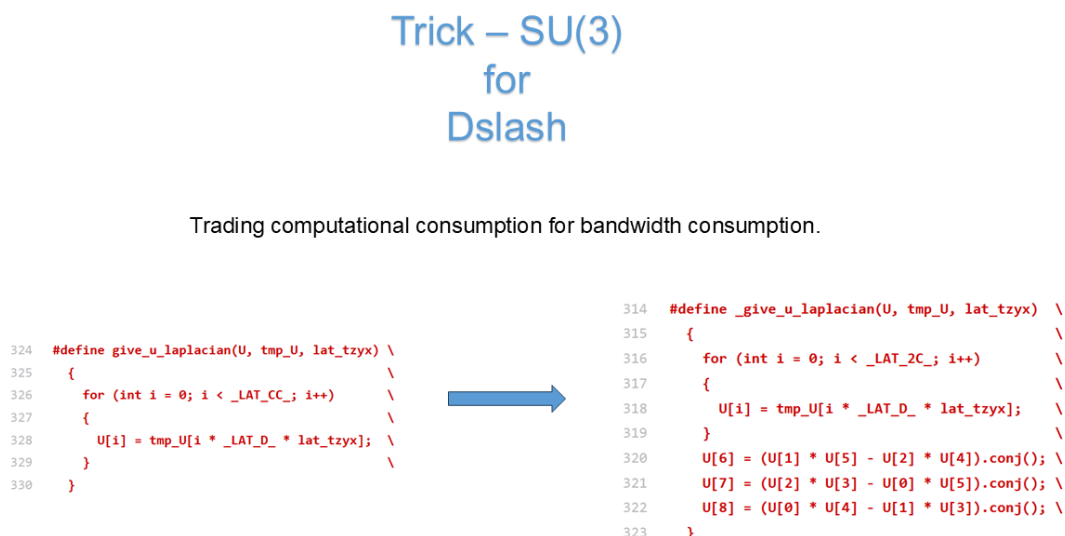
```
{
    // x+1
    move_forward_x(move, x, lat_x, eo, parity);
    tmp_u = (origin_u + {x, *_EVER_000 + parity} * lat_txyz);
    give_u(u, tmp_u, lat_txyz);
    tmp_src = {origin_src + move};
    get_src(src, tmp_src, lat_txyz);
}
{
    for (int c0 = 0; c0 < _LAT_C; c0++)
    {
        tmp0 = zero;
        tmp1 = zero;
        for (int c1 = 0; c1 < _LAT_C; c1++)
        {
            tmp0 += (src[c1] - src[c1 + _LAT_3C].multi_i(dagger_val)) * U[c0 * _LAT_C + c1];
            tmp1 += (src[c1 + _LAT_1C] - src[c1 + _LAT_2C].multi_i(dagger_val)) * U[c0 * _LAT_C + c1];
        }
        dest[c0] += tmp0;
        dest[c0 + _LAT_3C] += tmp1;
        dest[c0 + _LAT_1C] += tmp1.multi_i(dagger_val);
        dest[c0 + _LAT_2C] += tmp0.multi_i(dagger_val);
    }
}
```

图表 13 优化 1 示意图

以其中的  $1 - \gamma_0$  为例，其矩阵形状为  $4 \times 4$ ，可见第一行乘以  $i$  的结果与第四行相同，同理第二行乘以  $i$  的结果与第三行相同。在实际编写代码时需要对这四行进行遍历，这时可以将第一行矩阵乘的结果乘以  $i$  后作为第四行的结果，第二行与第三行同理，以此可以减少几乎一半的计算消耗。

### 3.5.2 优化 2

在进行 Dslash 操作时，其颜色维度体现为  $SU(3)$  矩阵元，其形式满足如下图右半部分所示的关系：



图表 14 优化 2 示意图

通过如上图所示的操作，可以将 GPU 上三分之一全局内存到设备内存读写过程转换为计算过程，对于以硬件带宽为瓶颈的读写密集型程序有着相当不错的性能提升。

### 3.5.3 优化 3

对于求解线性方程组的通用算法（例如上述的 CG 算法与 BISTABCG 算法）有一种巧妙的优化手段——奇偶预处理。其大体思路为将求解一个线性方程组问题中的矩阵分为奇偶两部分，通过分块后的性质得到两个互相关的线性方程组，再求解其中一个线性方程组来得到一个部分的解，代入另外一个线性方程组得到另外一个解，最后合并两个部分解得到完整解。较为详细的推导如下图所示：

## Trick - even-odd preprocessing for Inverse

Although the computational consumption of a single loop is unchanged, the memory occupation is reduced to half of the original, the overall convergence times are reduced, and the efficiency is significantly improved.

$$A_{ee}x_e - \kappa D_{eo}x_o = b_e$$

$$(A_{oo} - \kappa^2 D_{oe} A_{ee}^{-1} D_{eo})x_o = \kappa D_{oe} A_{ee}^{-1} b_e + b_o$$

origin one :  $\mathcal{M}x = b$

$$\begin{pmatrix} 1 - \kappa T_{ee} & -\kappa D_{eo} \\ -\kappa D_{oe} & 1 - \kappa T_{oo} \end{pmatrix} \begin{pmatrix} x_e \\ x_o \end{pmatrix} = \begin{pmatrix} b_e \\ b_o \end{pmatrix}$$

$$A = 1 + T$$

in this case,  $T = 0$ , so  $A = 1$

so,

$$\begin{pmatrix} 1 & -\kappa D_{eo} \\ -\kappa D_{oe} & 1 \end{pmatrix} \begin{pmatrix} x_e \\ x_o \end{pmatrix} = \begin{pmatrix} b_e \\ b_o \end{pmatrix}$$

$$x_e - \kappa D_{eo}x_o = b_e$$

$$x_o - \kappa D_{oe}x_e = b_o$$

so,

$$x_e - \kappa D_{eo}x_o = b_e$$

$$x_o - \kappa D_{oe}(\kappa D_{eo}x_o + b_e) = b_o$$

then

$$x_e - \kappa D_{eo}x_o = b_e$$

$$x_o - \kappa^2 D_{oe}(D_{eo}x_o) = \kappa D_{oe}b_e + b_o$$

so,

$$\text{just solve } x_o - \kappa D_{oe}(\kappa D_{eo}x_o) = \kappa D_{oe}b_e + b_o$$

$$\text{then will easily get } x_e \text{ by } x_e - \kappa D_{eo}x_o = b_e$$

so,

give Dslash :

$$tmp = D_{eo}src_o$$

$$dest_o = src_o - \kappa D_{oe}(\kappa D_{eo}src_o) = src_o - \kappa^2 D_{oe}tmp$$

give b:

$$b_e = anw_e - \kappa D_{eo}anw_o$$

$$b_o = anw_o - \kappa D_{oe}anw_e$$

$$b'_o = b_o + \kappa D_{oe}b_e$$

so,

$$Dslash(x_o) = b'_o$$

then get  $x_o$  by BistabCg,

This comes at the cost of a bit more overall code complexity.....

finally get  $x_e$  by  $b_e + \kappa D_{eo}x_o$

图表 15 优化 3 的示例图

实际运行按上述优化过的代码，我们发现单次循环的Dslash计算量并没有减少，但是由于矩阵大小变为原来的一半即求解线性方程组的规模变为原来的一半导致算法迭代收敛变快，所需的总的迭代次数变少，总体来说花费的时间减少。

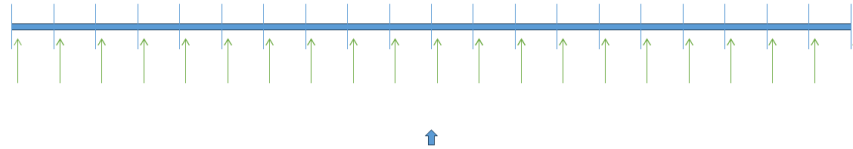
### 3.5.4 优化 4

对于CUDA程序有一种与硬件架构直接联系的优化方案——访存合并。如下图所示是没有经过访存合并处理过的读写过程：

V: lat\_t \* lat\_z \* lat\_y \* lat\_x  
SC: lat\_s \* lat\_c  
BLOCK\_SIZE: 128 or 256 and so on  
GridDim: V / BLOCK\_SIZE

## Trick - coalesced in cuda

[V(20)\*SC(12)]



The reads are too scattered and not efficient.

图表 16 优化 4 示意图（未优化）

可见硬件层面需要分多次读写数据，但实际上现在的GPU基本都支持一次读写多个相邻的数据，即访存合并，这要求编写程序时的多线程分配逻辑要与数据排列逻辑相对应。然而，一般来说多线程分配逻辑参考的是时空维度（例如 $32*32*32*64$ 的格子，其大小与常用GPU的CUDA核心数目相匹配，而其他维度显然太小了，无法充分发挥所有CUDA核心的性能），数据排列逻辑则按照朴实的物理认知将自旋维度（大小为4）与颜色维度（大小为3）排列在最后，这导致了两种逻辑不匹配，即不能访存合并。

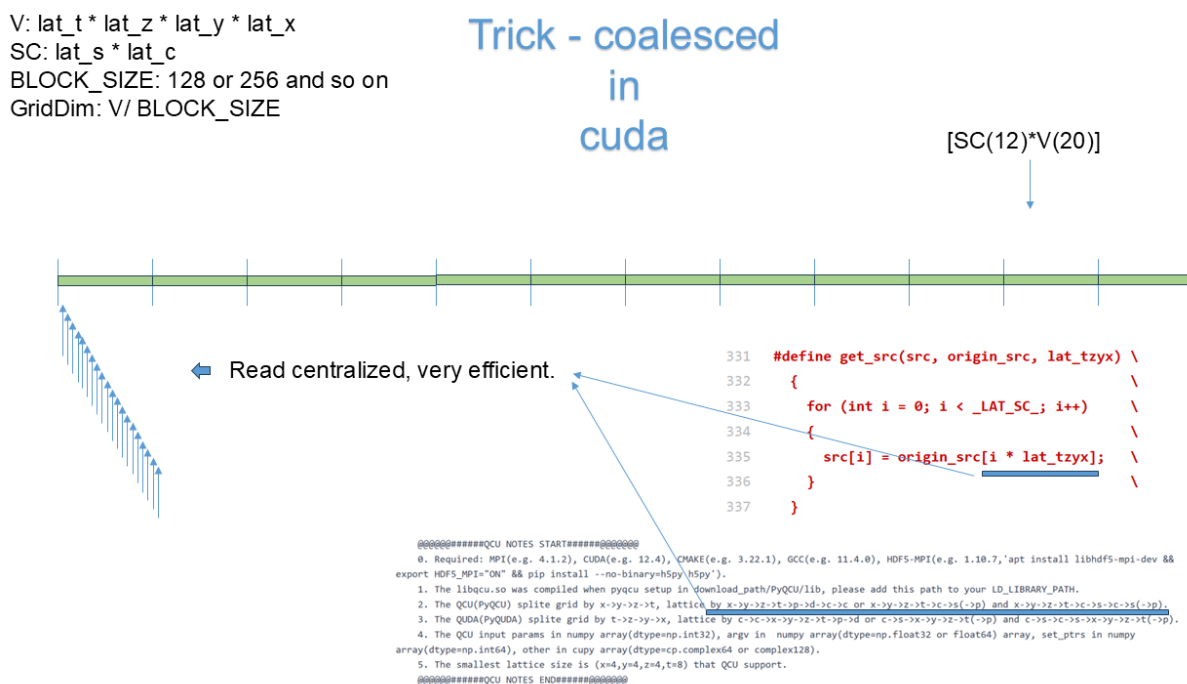
解决方案其实在之前的接口代码展示中已经给出了答案：

```
dptzyxcc2ccdptzyx < T > (gauge, &_set);
```

即在进行Dslash之前调换不同维度的顺序，将时空维度排布到相邻的位置（内存中的相邻位置，即最后几位的维度），在完成Dslash后再调换回去：

```
ccdptzyx2dptzyxcc < T > (gauge, &_set);
```

这时的读写过程如下图所示：



图表 17 优化 4 示意图（已优化）

## 第四章 结果产生与分析

### 4.1 产生与分析思路

为了排除个人消费级硬件的不稳定因素，本次测试结果在南方核科学计算中心（Southern Nuclear Science. Computing Center, SNSC）的gpu025节点（8张NVIDIA A100-

SXM-80GB) 上产生。为了验证结果的准确性并了解与国外同类型先进软件的差距, 本次测试在运行本课题程序的同时引入由NVIDIA资深工程师开发的QUDA程序作为对照组。QUDA历经15年的发展(2010年正式发布)且至今仍在积极地高强度更新, 并且常年作为NVIDIA公司高性能计算(HPC)的代表宣传, 是当之无愧的LQCD程序标杆。但QUDA有许多特性在本课题中并未实现, 所以为了与本课题程序(以下简称QCU)进行单一基础功能对比, 对QUDA(<https://github.com/lattice/quda/releases/tag/v1.1.0>)设置以下环境变量:

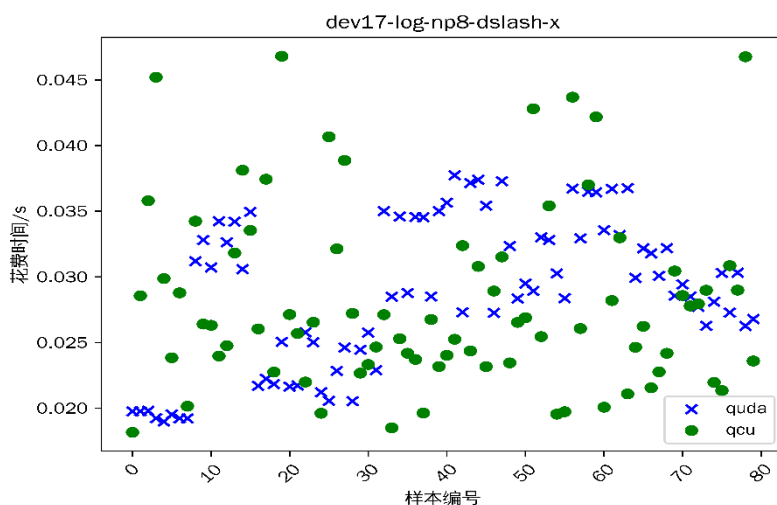
```
23  ## quda
24  export QUDA_ENABLE_P2P=0
25  export QUDA_ENABLE_TUNING=0
```

图表 18 QUDA 环境变量设置

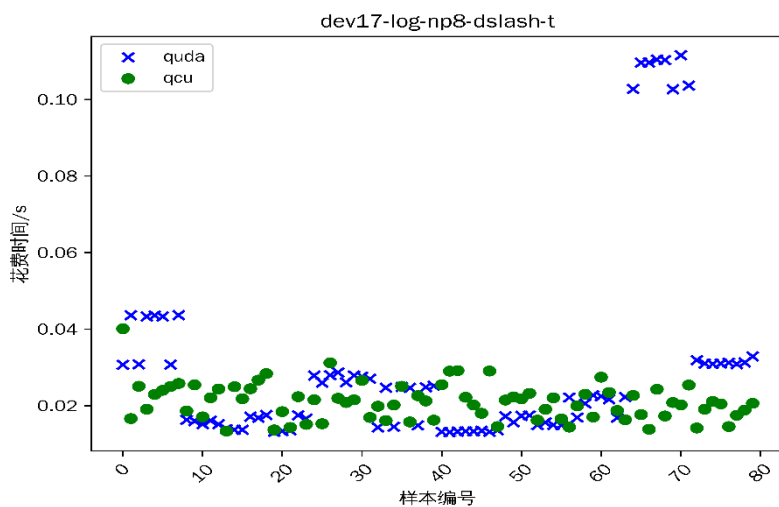
另外由于本次测试为八卡测试, 每一张计算卡都会分配一个MPI线程, 所以当运行样本数量为10时, 实际输出的耗时记录有 $8 \times 10 = 80$ 个, 后续图表中将有所体现, 最后格子大小统一为 $32 \times 32 \times 32 \times 64$  (X、Y、Z、T)。

## 4.2 Wilson Dslash 的结果

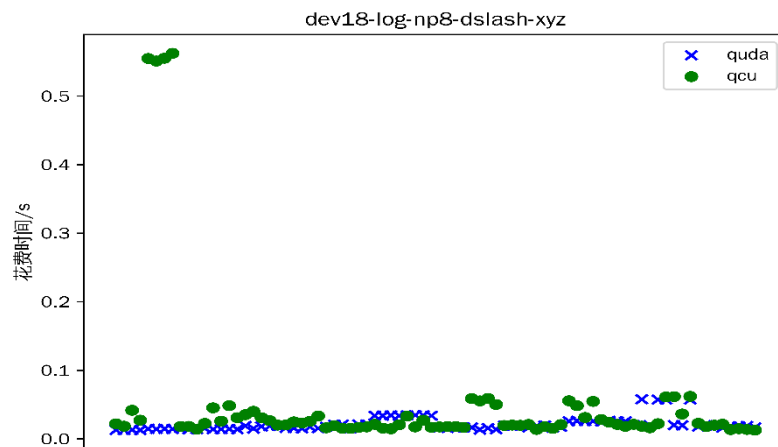
本项测试按照不同的多卡分割方式有4组数据, 每组数据有10个样本, 实际输出80个样本, 共计320个样本, 每个样本由一次Wilson Dslash的QCU耗时与QUDA耗时组成, 图表如下所示:



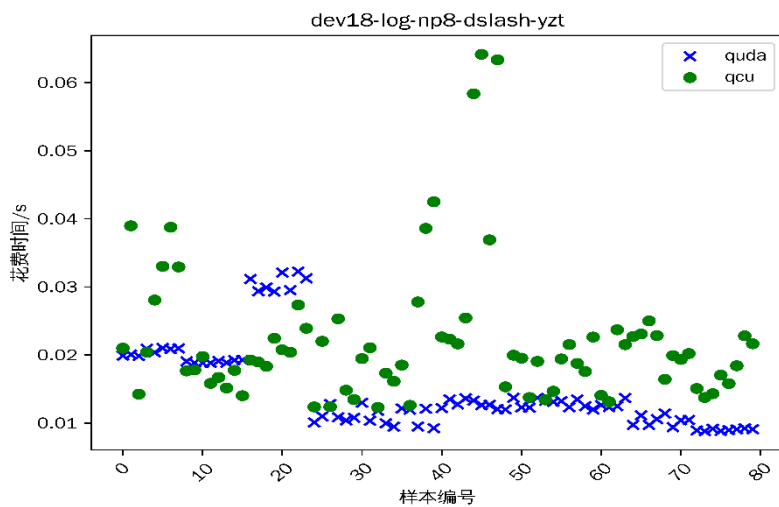
图表 19 4.2.1 测试结果图



图表 20 4.2.2 测试结果图



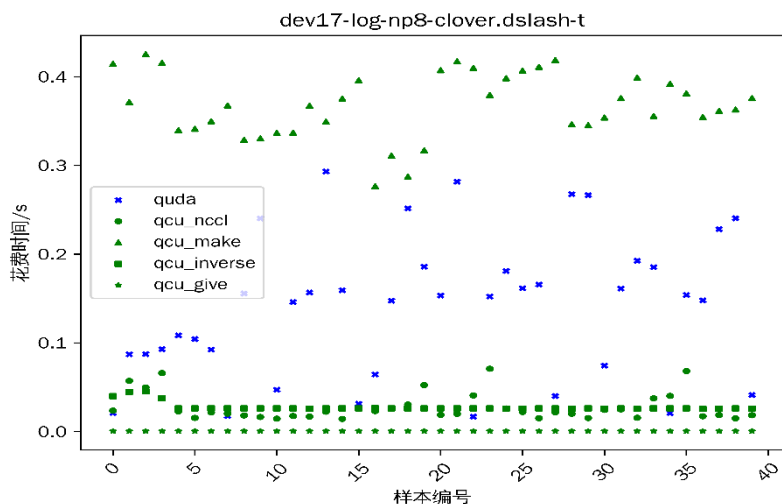
图表 21 4.2.3 测试结果图



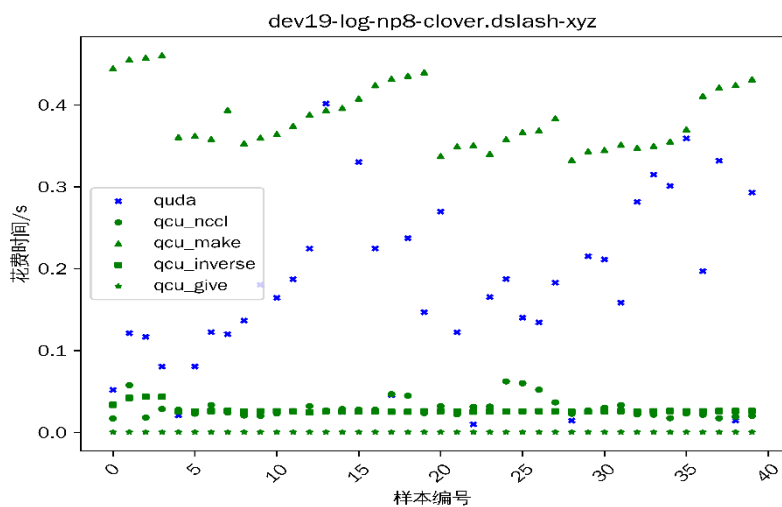
图表 22 4.2.4 测试结果图

### 4.3 Clover Dslash 的结果

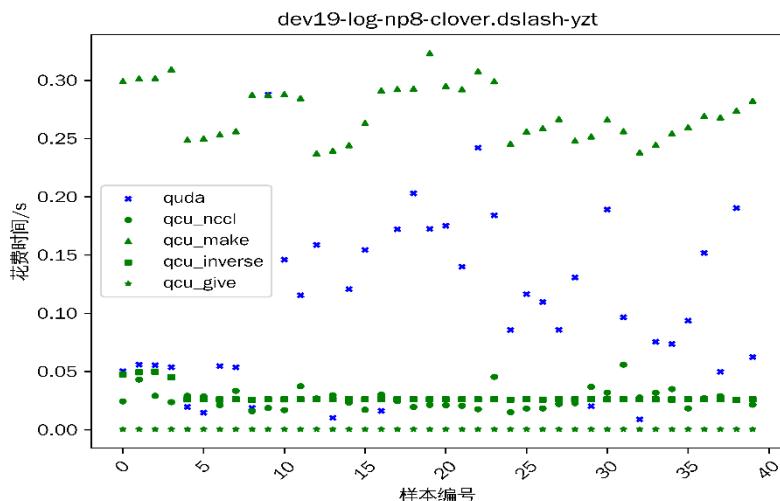
本项测试按照不同的多卡分割方式有3组数据，每组数据有5个样本，实际输出40个样本，共计120个样本，每个样本由一次Clover Dslash的QCU耗时(分为4部分)与QUDA耗时组成，图表如下所示：



图表 23 4.3.1 测试结果图



图表 24 4.3.2 测试结果图

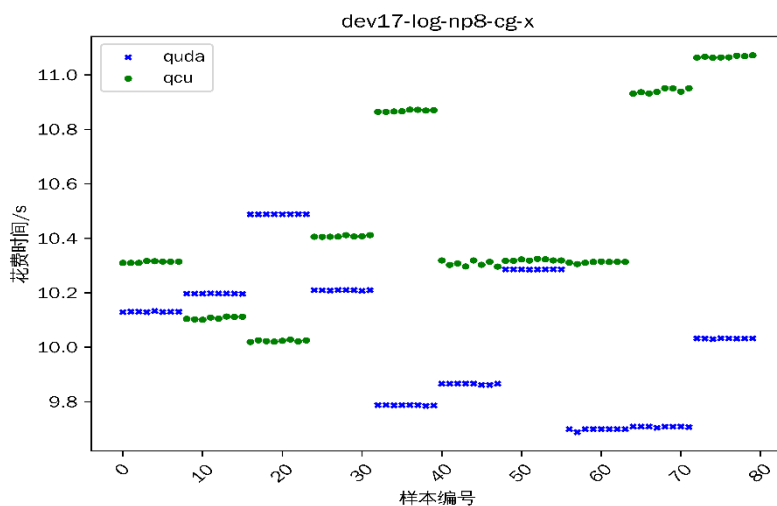


图表 25 4.3.3 测试结果图

## 4.4 求解算符的结果

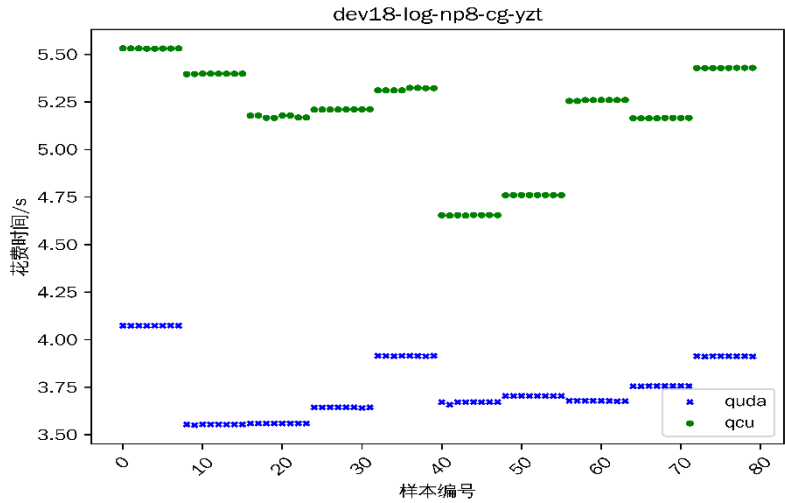
### 4.4.1 CG 算法的结果

本项测试按照不同的多卡分割方式有 3 组数据，每组数据有 10 个样本，实际输出 80 个样本，共计 240 个样本，每个样本由一次 CG 的 QCU 耗时与 QUDA 耗时组成，图表如下所示：

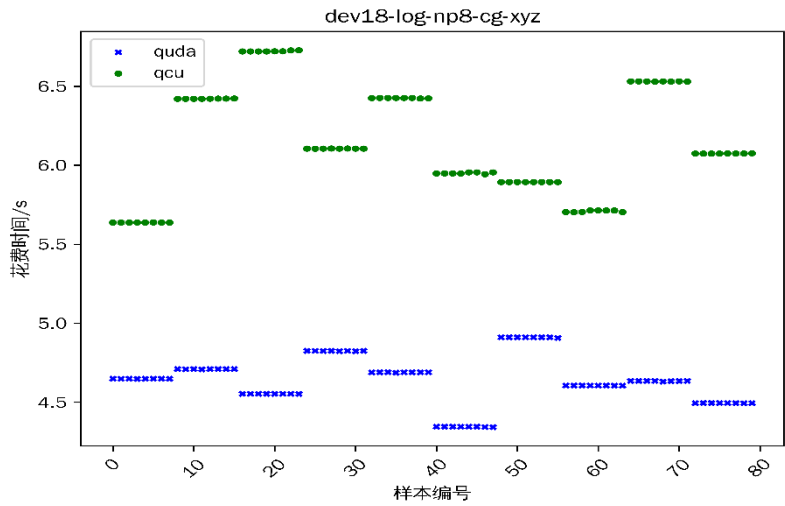


图表 26 4.4.1.1 测试结果图





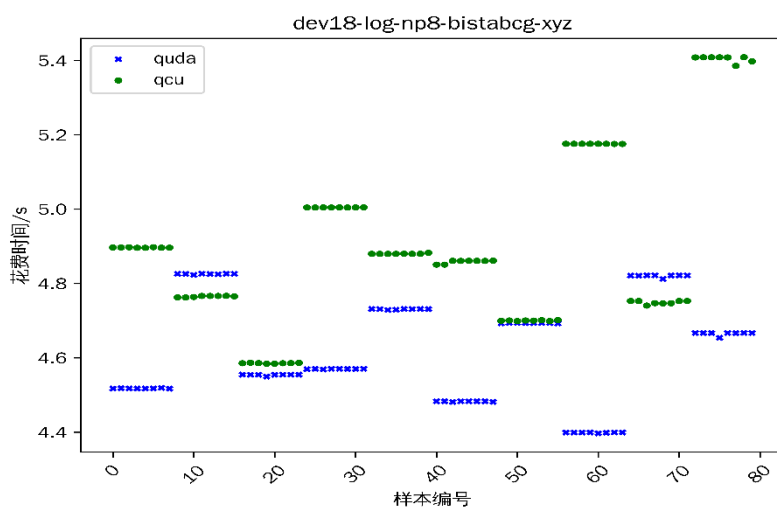
图表 27 4.4.1.2 测试结果图



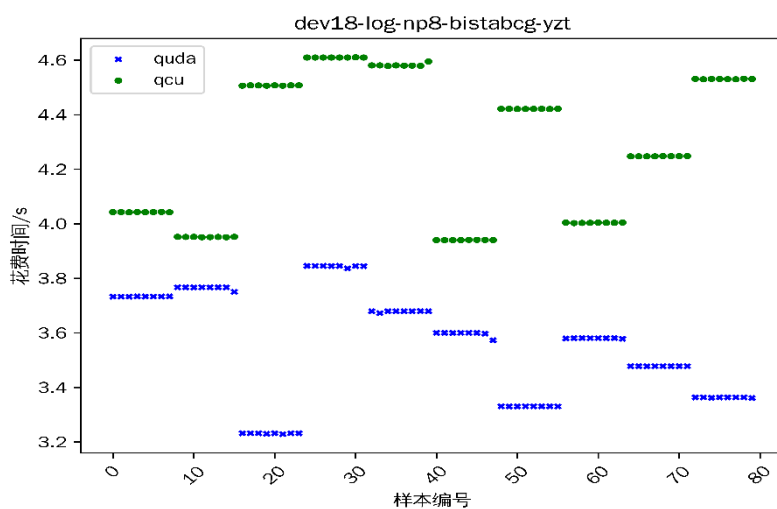
图表 28 4.4.1.3 测试结果图

4. 4. 2 BISTABCG 算法的结果

本项测试按照不同的多卡分割方式有2组数据，每组数据有10个样本，实际输出80个样本，共计160个样本，每个样本由一次BISTABCG的QCU耗时与QUDA耗时组成，图表如下所示



图表 29 4.4.2.1 测试结果图



图表 30 4.4.2.2 测试结果图

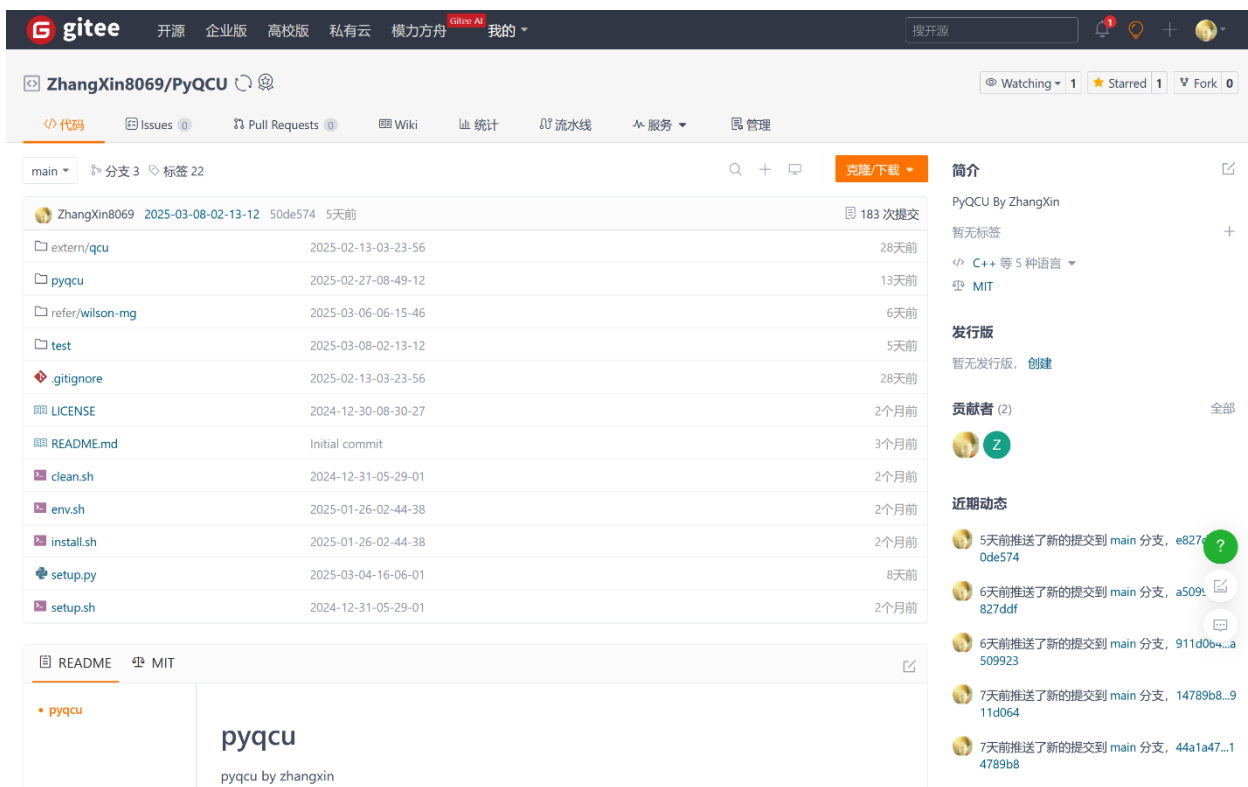
## 4.5 总体分析

综合上述的图表数据，QCU在Wilson Dslash、Clover Dslash、CG、 BISTABCG这四个求解传播子的基本功能中，已经达到了QUDA的70%-80%性能（特定版本与特定环境变量下，见4.1）。

## 第五章 用户配置

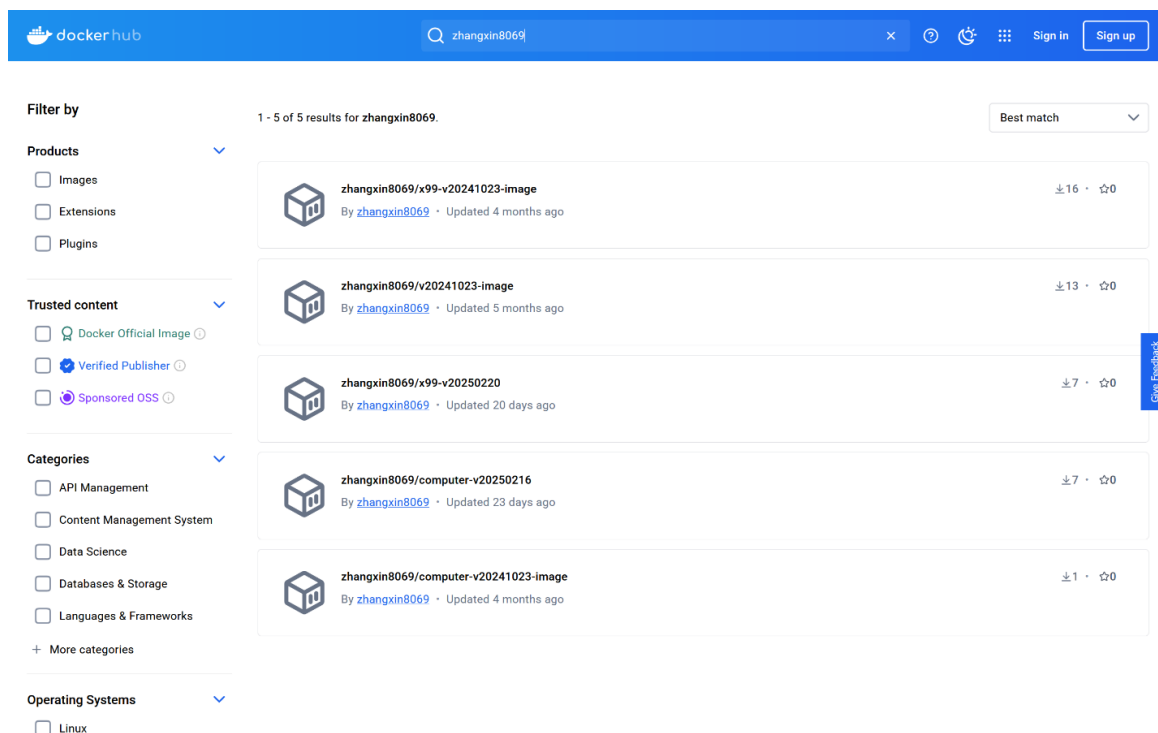
### 5.1 PyQCU

为了方便用户使用，本课题还构建了面向实际应用的Python计算接口（访问地址：<https://gitee.com/zhangxin8069/PyQCU>），为后续大规模物理问题求解奠定了完整的软件基础。



## 5.2 Docker

考虑到未来的用户上手Linux可能有一定的困难（主要是系统环境配置），在此提供了开箱即用的Docker镜像：



## 第六章 结论与展望

本课题总体来说是一项解决格点量子色动力学中的热点计算问题即部分子分布函数中的传播子求解问题的国产化工作。区别于直接参考国外先进软件来做逆向工作，本课题从格点场论的理论出发并借鉴了一些公知的优秀的算法优化方案，从无到有复现了前述的先进软件的一些关键功能。虽然这项工作在当下因为种种原因并不能改变我国依赖国外软件的现状，例如：格点的实际研究依赖相当庞大的定制化软件生态，本课题实现的只是其中的牛毛，并没有实现一套完整的工作流；即使本课题为其它软件做了相同的接口，因为与原软件有着不可忽略的性能差距，进行替换并没有实际的经济意义。但是如果我们着眼于未来，这项工作对于后续我国格点的自主发展具有相当的必要性，这主要体现在以下几个方面：随着贸易壁垒日益加剧，国产高性能计算硬件势在必行，而国外开发者并不会着力于适配尚且弱势的国产硬件，我们需要软件与硬件协同发展，本课题正是在为此做前期准备；实际格点工作中我们会产生一些独特的想法，而这些想法落地需要对应的独特的软件实现，本课题依靠对自身的完整掌握可以为此提供一定的软件支持。

更进一步，本课题截至本文定稿已经完成了立项时定下的基本目标——使用完全自研自产的国产代码实现国外先进软件的基础功能，同时对于前文未提到的进阶目标即实现进阶功能——MultiGrid算法正在理论攻坚环节。此进阶目标难度巨大（上述功能的总代码量大概为15000行，而MultiGrid算法的总代码可能近50000行），但是一旦完成就具有相当大的投入实际工作的可能性，望我们接下来进展顺利，早日为中国格点做出实际的贡献。

## 参考文献

- [1] 刘川. 格点量子色动力学导论. 北京大学出版社, 1 edition, 6 2017.
- [2] 蒋翔宇. 轻强子性质的格点 QCD 研究. PhD thesis, 中国科学院高能物理研究所, 2023.
- [3] Xiangdong Ji. Parton physics from large-momentum effective field theory. *Science China Physics, Mechanics & Astronomy*, 57:1407–1412, 2014.
- [4] Fei Yao, Lisa Walter, Jiunn-Wei Chen, Jun Hua, Xiangdong Ji, Luchang Jin, Sebastian Lahrtz, Lingquan Ma, Protick Mohanta, Andreas Schafer, et al. Nucleon transversity distribution in the continuum and physical mass limit from lattice qcd. *Physical Review Letters*, 131(26):261901, 2023.

## 致 谢

本课题萌芽于本人大一暑假末到大二开学初的那段时间（2022年秋）。当本人的导师即中国科学近代物理研究所（以下简称近物所）的孙鹏研究员交给我一段LQCD代码（CPU单线程的二维Wilson Dslash示例）的复现工作时，本人第一次在大学感到了深切的无力感——上大学后才接触电脑、物理课程仅仅到电磁学、听名字就很难的格点量子色动力学（实际上确实很难……）。但在一次次的孙老师从那时就开始的每周六组会中，本人竟然一步步地坚持了下来，直到定稿的当下依旧如常。本人感叹于虽岁月蹉跎但润物无声（本人这样的浑浑噩噩过日子的平凡本科生经过两年多的磨练后竟也成为了一个毛细领域的小小专家），更钦佩于孙老师的耐心与包容，在此最先真诚地表示感谢。同时也非常感谢CLQCD（中国格点组，China Lattice QCD Collaboration）组内各位老师同学，诸位的帮助让本人真切地感受到中国格点人的温暖，也让本人更加相信中国格点在未来会有更加长足的发展，这也是本人研究生方向依旧坚持格点的重要原因。另外也感谢于国内外的开源作者，你们提供的开源的软件与教程是本课题的基石之一。本人从开源中学习了許多，也从开源中感受到了难能可贵的精神——人人为我，我为人人！本人真诚地希望世界能够拥抱开源、享受开源、习惯开源。另外感谢本人的校内指导老师即于福升教授，您对于物理世界的热爱令人鼓舞，您对于本课题的帮助更令本人受宠若惊，在此再次感谢。最后感谢学院与近物所的教导与支持，真诚地希望学院与近物所欣欣向荣，也希望本人在日后能够带回成果，借此来回报此栽培之恩。