High-Performance Implementation of Multigrid Solver for Lattice QCD

中国科学院近代物理研究所
张鑫
合作者：孙鹏 张克龙 杨冀翀

第六届中国格点量子色动力学研讨会
香港中文大学(深圳)
2026年10月11日

目录

一. PyQCU的简要背景

二. PyQCU的求解对象

三. MultiGrid算法简介

四. PyQCU对比QUDA

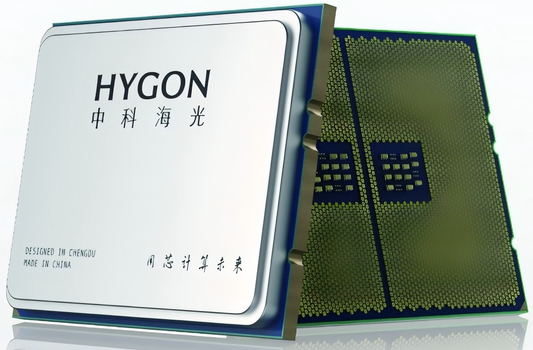
1. 架构与算法
2. 实战性能图

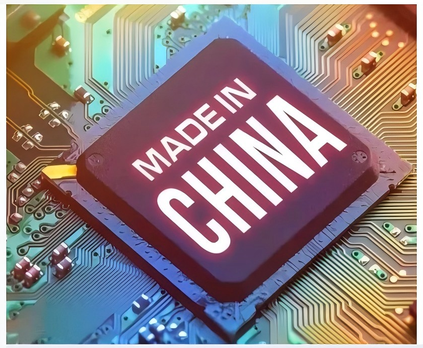
五. PyQCU在国产计算平台

六. 总结与展望







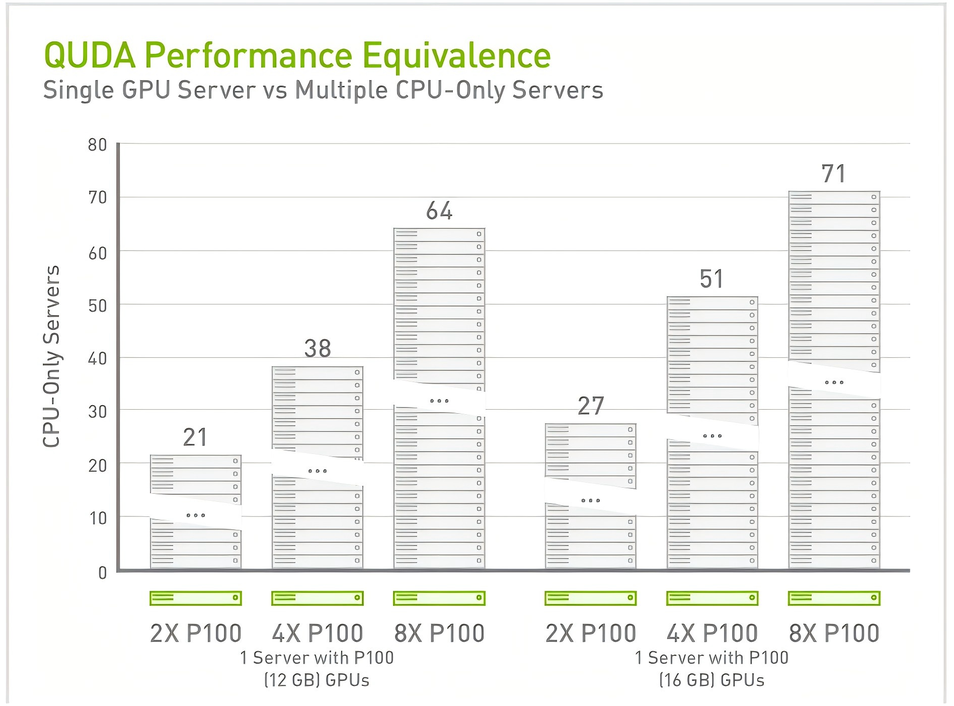






一、 PyQCU的简要背景

求解传播子是当前格点QCD工作中不可忽视的**计算热点**，所幸NVIDIA公司的开源项目**QUDA**已经对此问题给出了非常优秀的高性能解决方案，但不幸的是这个项目在向各个**国产计算平台**移植时出现了诸多困难，难以发挥这些优秀国产计算资源的全部实力，为了更加充分地利用国产计算卡并且给日后自主解决其他新的计算热点做铺垫，我们启动了开源项目**PyQCU**，截至当下已经实现了核心的且有较高性能的**MultiGrid求解器**



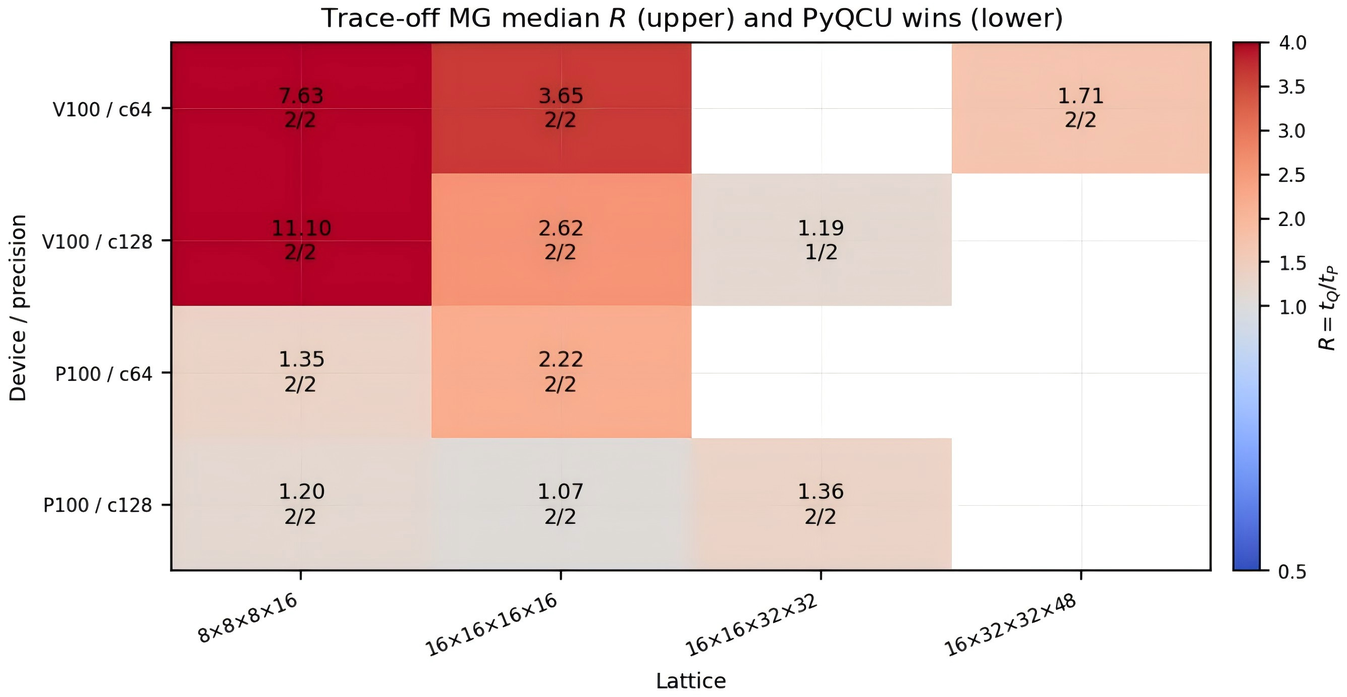
QUDA
A library for Lattice Quantum Chromo Dynamics on GPUs

VERSION
2017

ACCELERATED FEATURES
All

SCALABILITY
Multi-GPU and Multi-Node

MORE INFORMATION
www.nvidia.com/quda



二、 PyQCU的求解对象

在QCD中拉氏量为：

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

其中ψ_i为夸克场，γ^{μ}为狄拉克矩阵，D_{μ}为协变导数，$F^{a}_{\mu\nu}$为规范胶子场张量

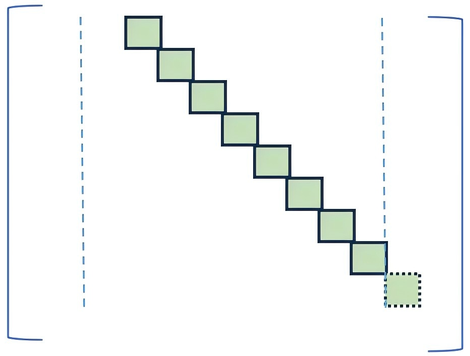
上式离散化后给出较为粗略的Wilson Dslash：

$$M^{W}_{x,\,y}[U]a = \delta_{xy} - \kappa\sum_{\mu}\left[(r-\gamma_{\mu})U_{x,\,\mu}\delta_{x,\,y-\mu} + (r+\gamma_{\mu})U^{\dagger}_{x-\mu,\,\mu}\delta_{x,\,y+\mu}\right]$$

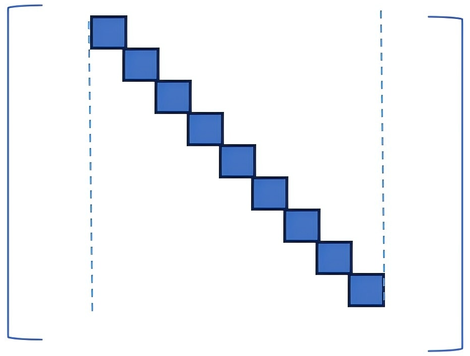
上式进一步修正后给出较为精确的LQCD主要使用的Clover Dslash：

$$M^{W}_{x,\,y}[U]a = \delta_{xy} - \kappa\sum_{\mu}\left[(r-\gamma_{\mu})U_{x,\,\mu}\delta_{x,\,y-\mu} + (r+\gamma_{\mu})U^{\dagger}_{x-\mu,\,\mu}\delta_{x,\,y+\mu}\right] - \frac{iaC_{SW}\kappa r}{4}\sigma_{\mu\nu}F_{\mu\nu}\delta_{x,\,y}$$

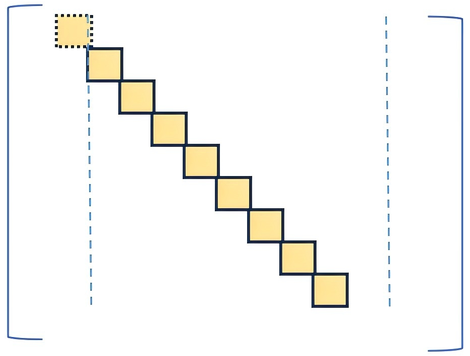
Wilson minus term



Clover & I term

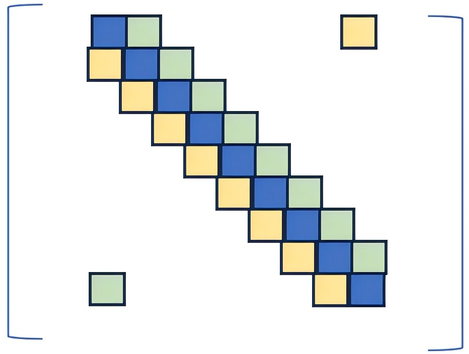


Wilson plus term



A * x = Ax

Whole Dslash Fermion Src Fermion Dest



*



=



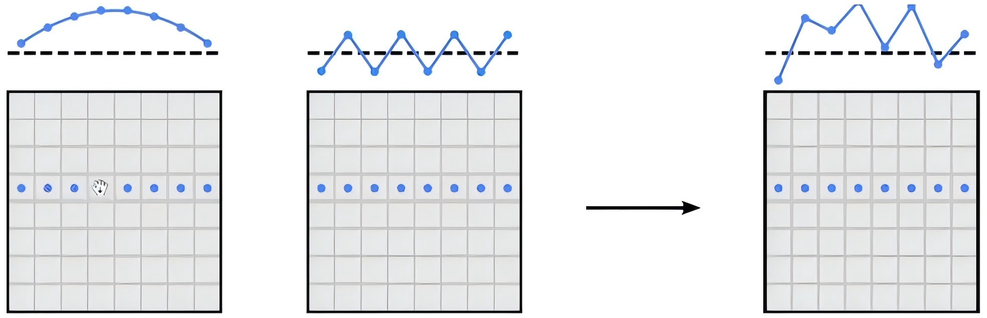
对前式中的M求逆的结果正是格点QCD中的传播子S，且此过程可以抽象为求解一个线性方程组，即$\boldsymbol{Ax = b}$，$\boldsymbol{x = A^{-1}b}$，其中$\boldsymbol{A}$为一个超大规模复数稀疏矩阵

三、 MultiGrid算法简介

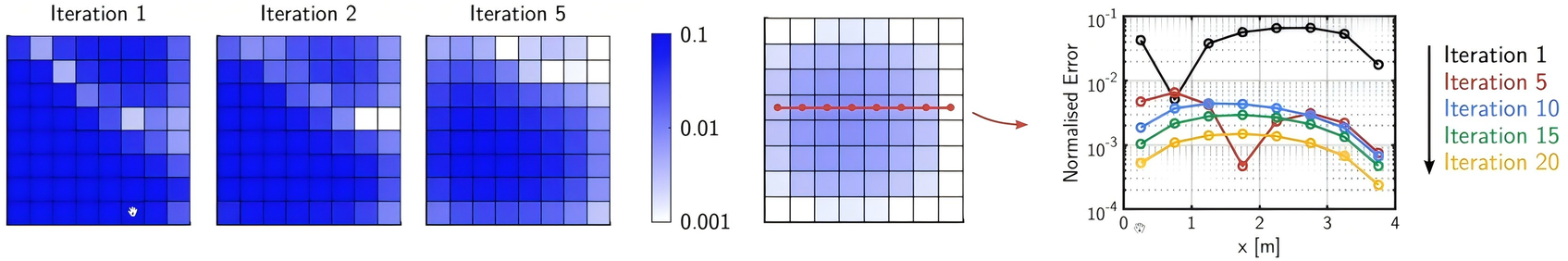
在一般的求解线性方程组的算法（以 BiCGStab为例，即 Biconjugate gradient stabilized method）中，每一次迭代都有：

$$e_i = x_i - x_{final}$$

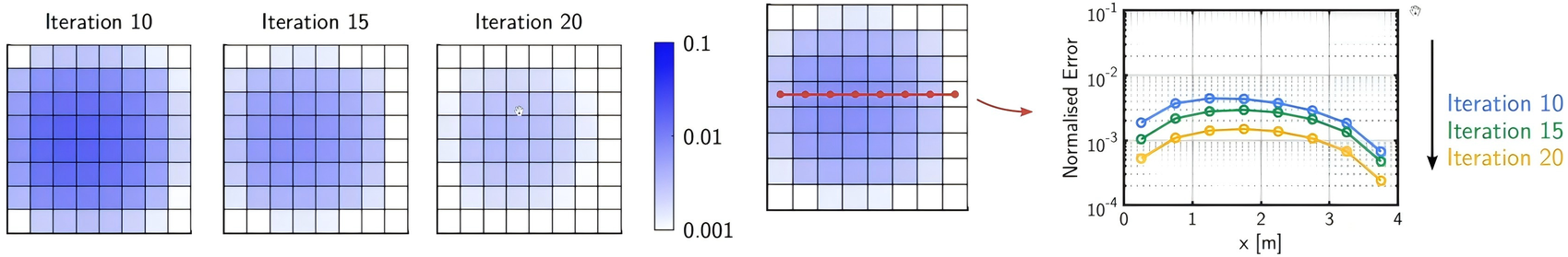
其中 $Ax_{final} = b$， e_i由所谓的**高频**部分与**低频**部分组成，如右图：



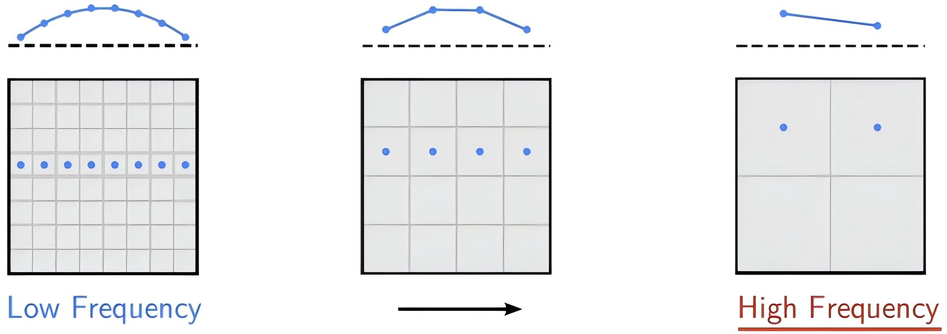
初始迭代的几个e_i中高频误差比较显著，而一般的迭代算法又对**高频误差敏感**，所以前几次迭代e_i（这里大小指模）会下降较快：



后续迭代的e_i中的高频误差被之前的迭代磨平，即转为低频误差，所以低频误差的占比逐渐变大，而一般的迭代算法又对**低频误差相对不敏感**，所以后续迭代的e_i 会下降**较慢**：



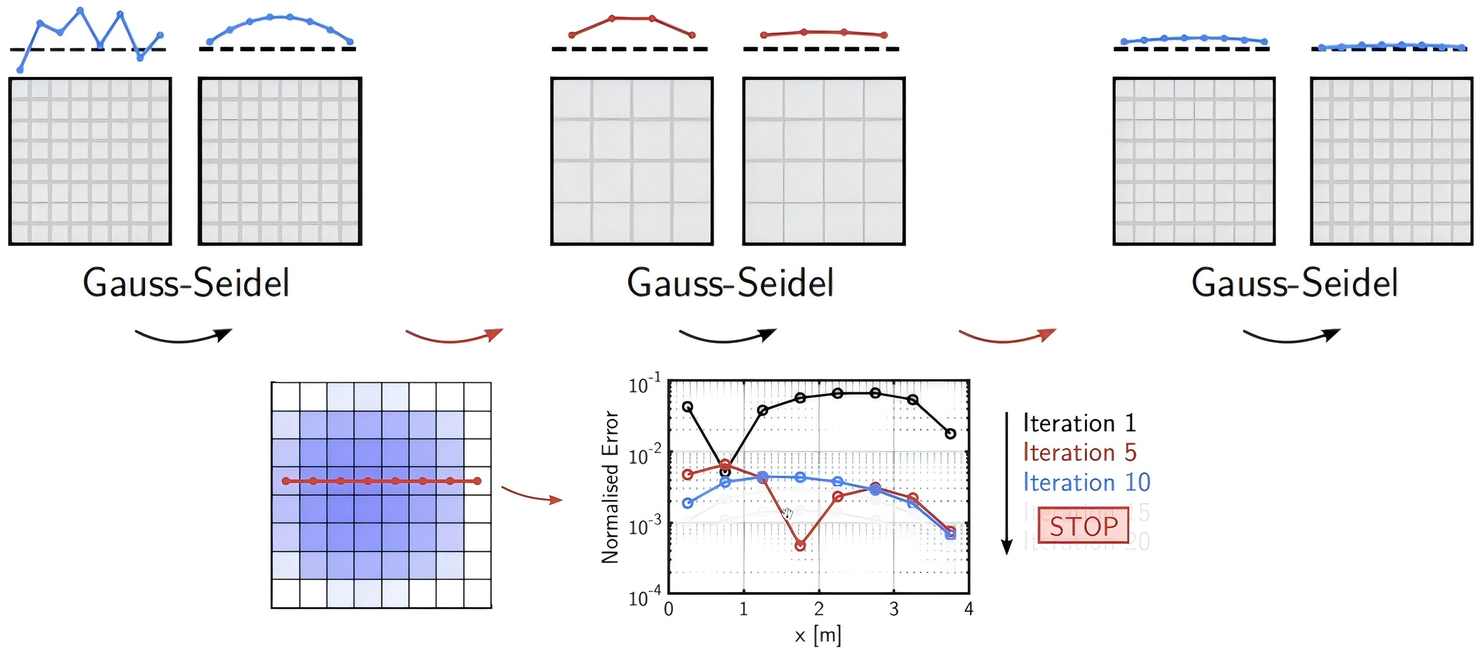
为了解决上述问题，MultiGrid算法引入了关键的“粗化”操作，后续迭代的e_i中的低频误差被更粗层的粗化操作转化为高频误差，再被更粗层的平滑器磨平：

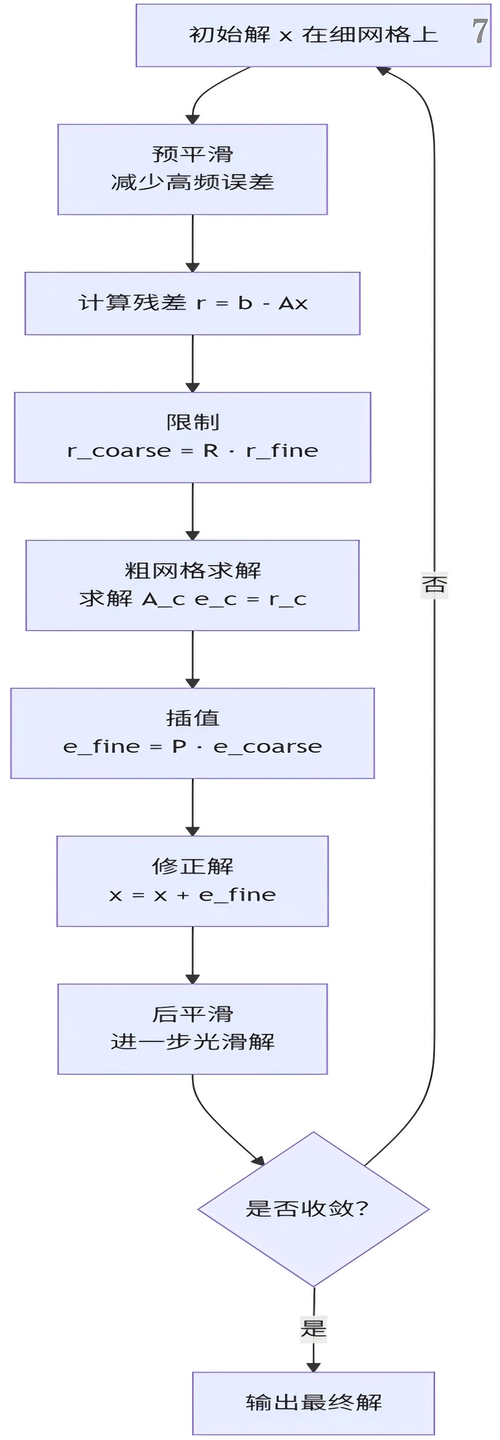


更进一步，一个简易的MultiGrid算法（双层）的实现步骤为：

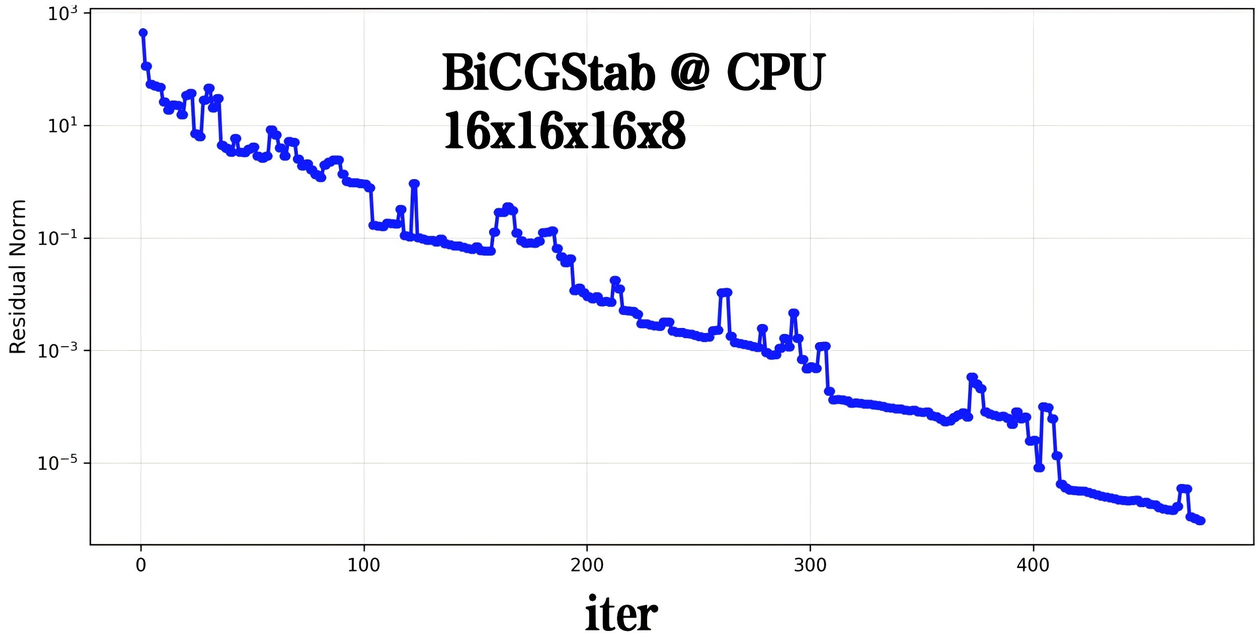
细层前平滑->残差粗化->粗层求解->误差细化->细层后平滑……

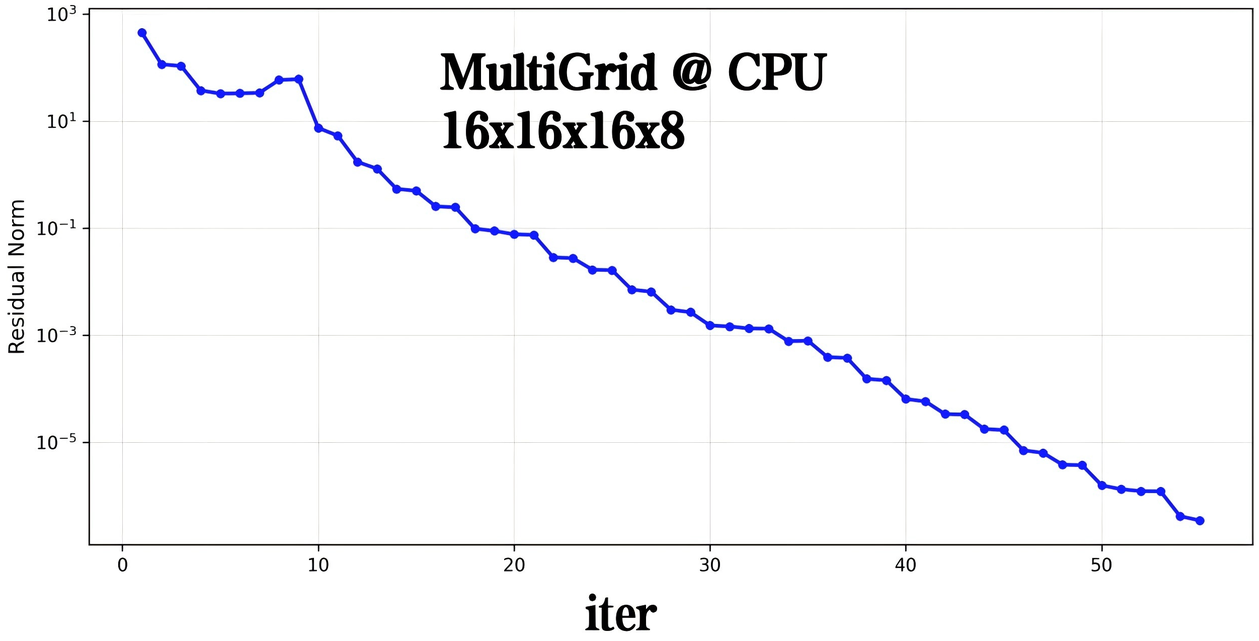
实际应用场景会有多层网格，并且对于当下层是继续粗化还是细化有特定的规则（如V-Cycle、W-Cycle、F-Cycle、FMG-Cycle等），在此右侧给出常用的V-Cycle：





MultiGrid算法显著减少迭代次数的效果示意图



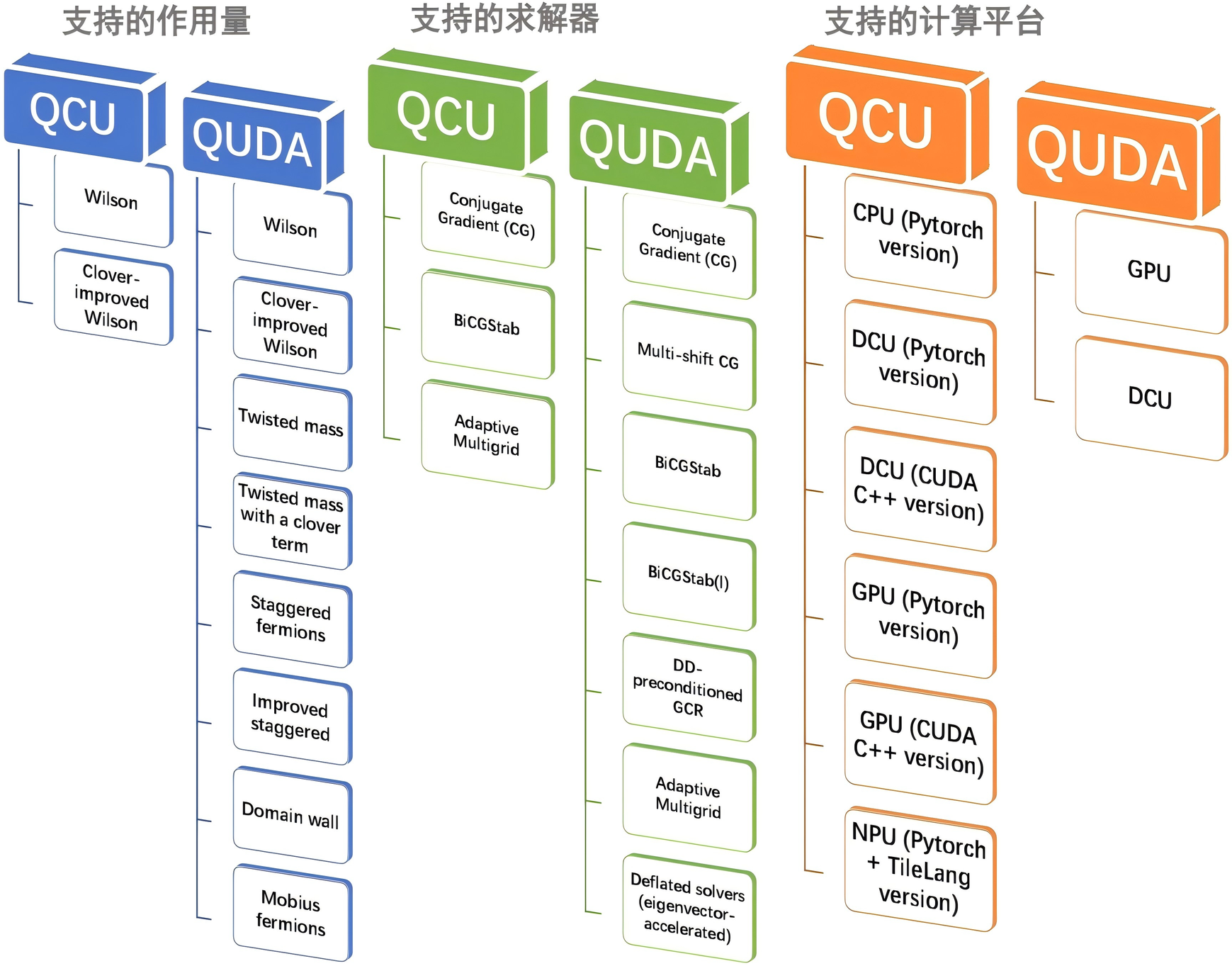


四、 PyQCU与QUDA的对比

1. 架构与算法

QUDA十分**成熟**，作用量与求解器支持**覆盖很广**，对NVIDIA平台极致优化，但也因如此，QUDA的代码复杂度相当高，向CUDA生态外移植难度较大，整体**广且整**

PyQCU目前专注于使用**特定**算法求解**特定**作用量下的传播子，代码从零构建，出发思路与QUDA存在较大差异，整体**窄但深**



PyQCU

full RHS $[2, 4, 3, X, Y, Z, T/2]$ ← fine prepare 归一化 odd MATPC ← restarted right-FGMRES requested restart 16 → full solution

preconditioner M^{-1}: each outer FGMRES iteration

fine MR pre $\nu_{\text{pre}} = 1$ → restriction R to full coarse → recursive V-cycle per-level MR → coarsest BiCGStab single-rank fused multi-rank distributed → prolongation P to fine correction → fine MR post $\nu_{\text{post}} = 1$ → true MATPC residual refresh

MultiGrid算法在PyQCU与QUDA中的实现对照图

QUDA

outer GCR effective restart 16/4 → DiracCloverPC odd–odd MATPC → Transfer aggregate P/R → DiracCoarse/PC $X/Y/\widehat{Y}$ → MR smoother $\nu_{\text{pre}}, \nu_{\text{post}}$ → lower-level wrapper cycle type → coarsest CA-GCR maxiter 200 → prepare / reconstruct full solution

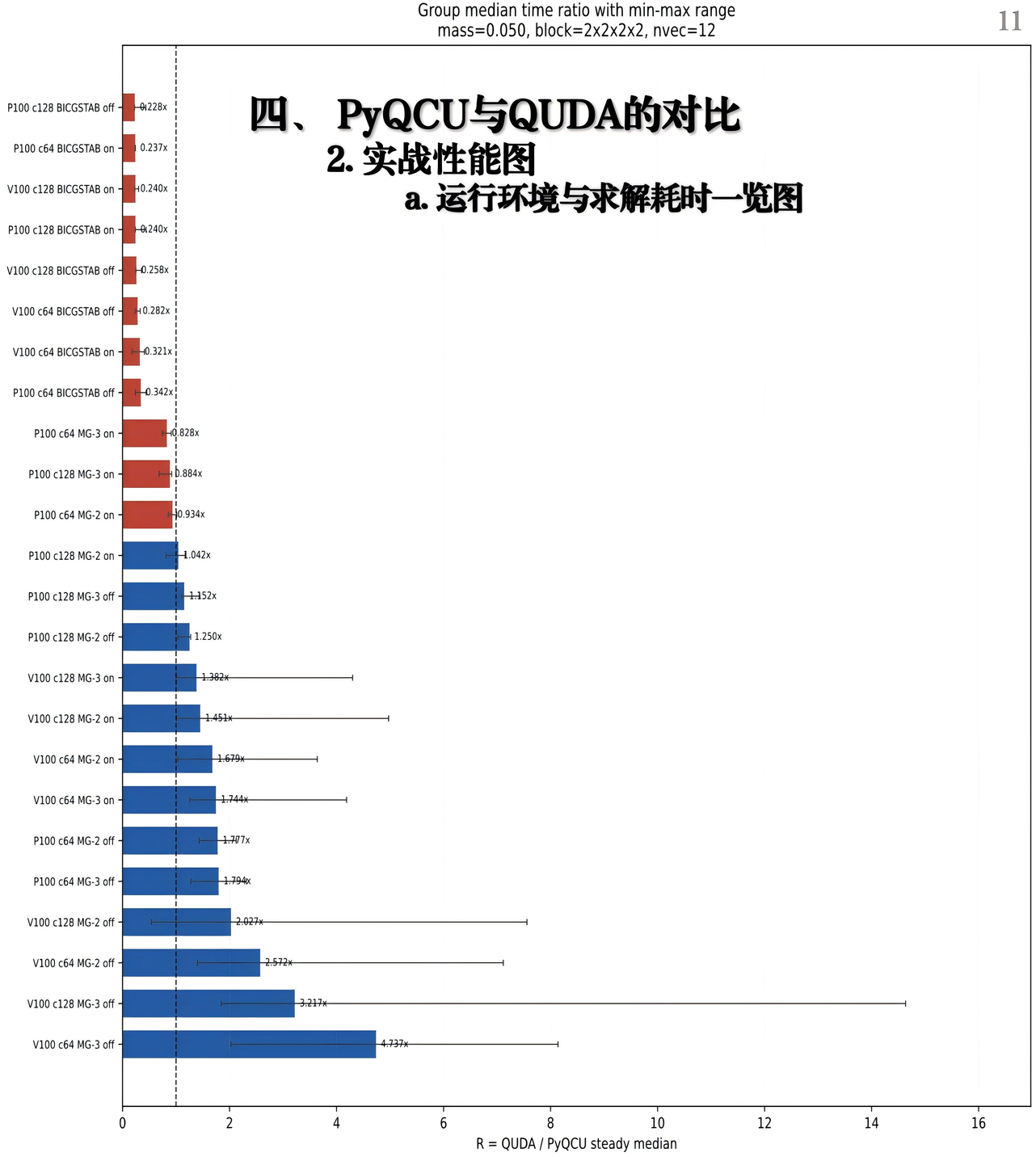
四、 PyQCU与QUDA的对比

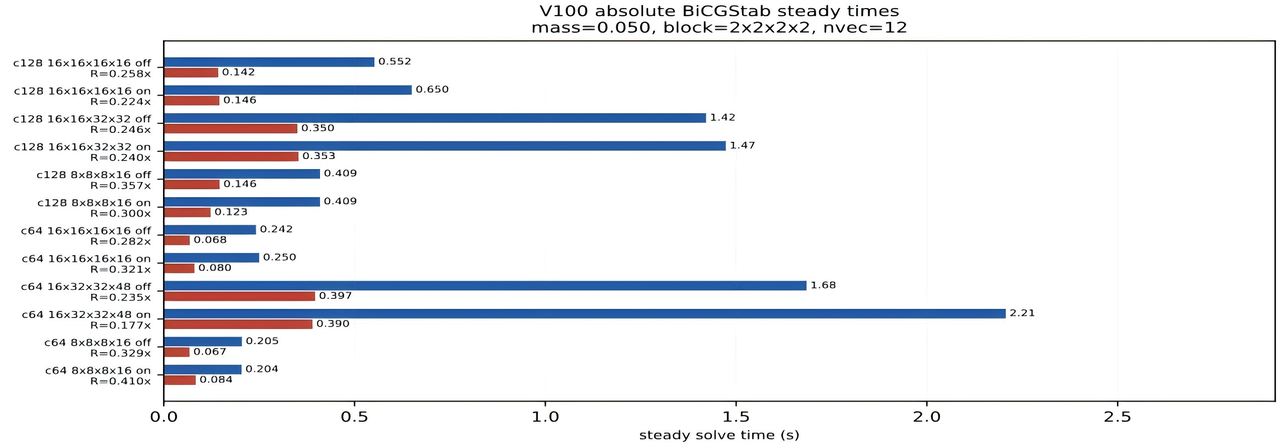
2. 实战性能图

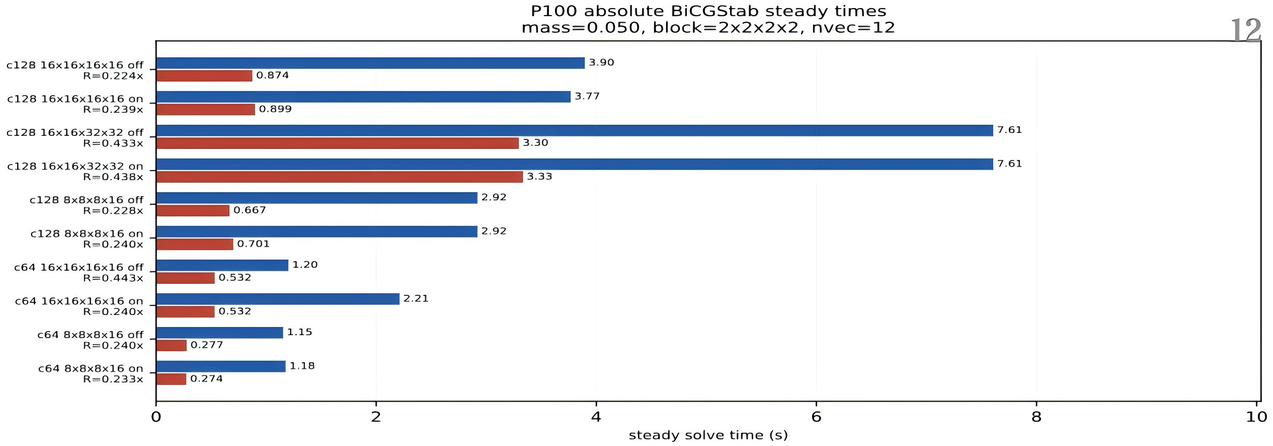
a. 运行环境与求解耗时一览图

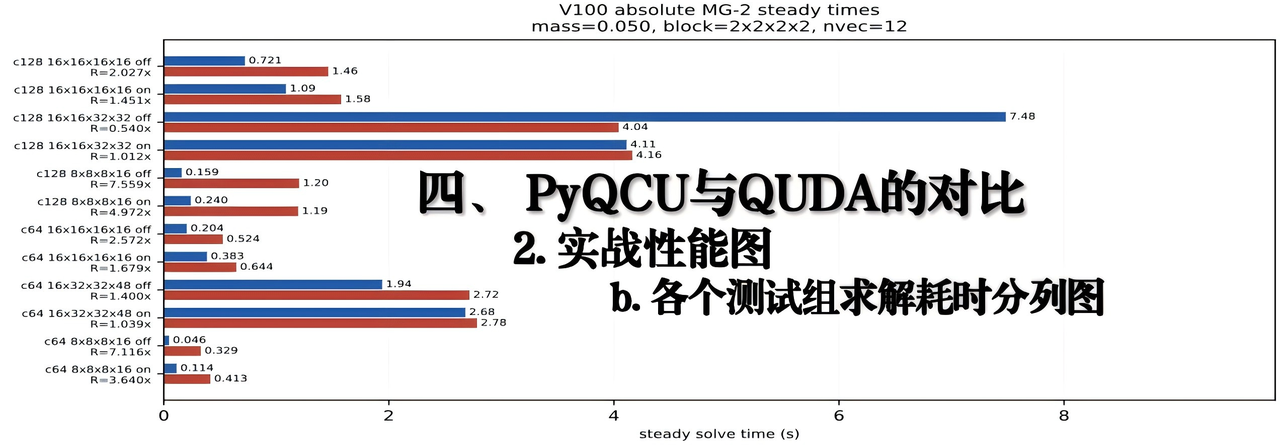
维度	取值
实现	QUDA 1.1.0，PyQCU Strict CUDA/C++
设备	单卡 Tesla V100-SXM2-32GB；双 Tesla P100-PCIE-16GB
格点	$8^3 \times 16$，16^4，$16 \times 32 \times 32 \times 48$，$16 \times 16 \times 32 \times 32$
精度	c64，c128
trace	on，off
求解	BiCGStab 参考，MG-2，MG-3
phase	1 cold，2 warmup，5 steady
统计	steady median；warmup 不进入性能统计
测试元数据	mass 0.050，kappa 0.1234567901234568，block $2 \times 2 \times 2 \times 2$，nvec 12

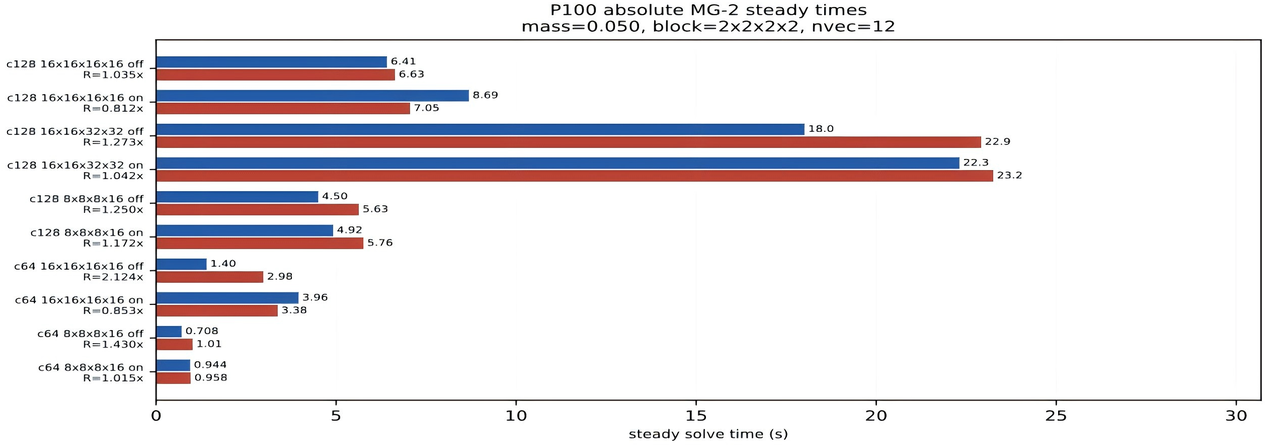
设备	精度	求解	trace	n	PyQCU 中位 s	QUDA 中位 s	R 中位
P100	c128	BICGSTAB	off	3	3.896	0.874	0.2284
P100	c128	BICGSTAB	on	3	3.767	0.899	0.2400
P100	c128	MG-2	off	3	6.410	6.634	1.2502
P100	c128	MG-2	on	3	8.686	7.052	1.0420
P100	c128	MG-3	off	3	4.688	5.185	1.1517
P100	c128	MG-3	on	3	8.364	5.713	0.8843
P100	c64	BICGSTAB	off	2	1.178	0.405	0.3416
P100	c64	BICGSTAB	on	2	1.694	0.403	0.2368
P100	c64	MG-2	off	2	1.055	1.995	1.7773
P100	c64	MG-2	on	2	2.450	2.166	0.9341
P100	c64	MG-3	off	2	1.073	2.057	1.7944
P100	c64	MG-3	on	2	2.894	2.263	0.8282
V100	c128	BICGSTAB	off	3	0.552	0.146	0.2580
V100	c128	BICGSTAB	on	3	0.650	0.146	0.2397
V100	c128	MG-2	off	3	0.721	1.461	2.0269
V100	c128	MG-2	on	3	1.085	1.575	1.4515
V100	c128	MG-3	off	3	0.436	1.525	3.2174
V100	c128	MG-3	on	3	1.173	1.621	1.3822
V100	c64	BICGSTAB	off	3	0.242	0.068	0.2818
V100	c64	BICGSTAB	on	3	0.250	0.084	0.3213
V100	c64	MG-2	off	3	0.204	0.524	2.5717
V100	c64	MG-2	on	3	0.383	0.644	1.6785
V100	c64	MG-3	off	3	0.151	0.715	4.7366
V100	c64	MG-3	on	3	0.528	0.921	1.7444







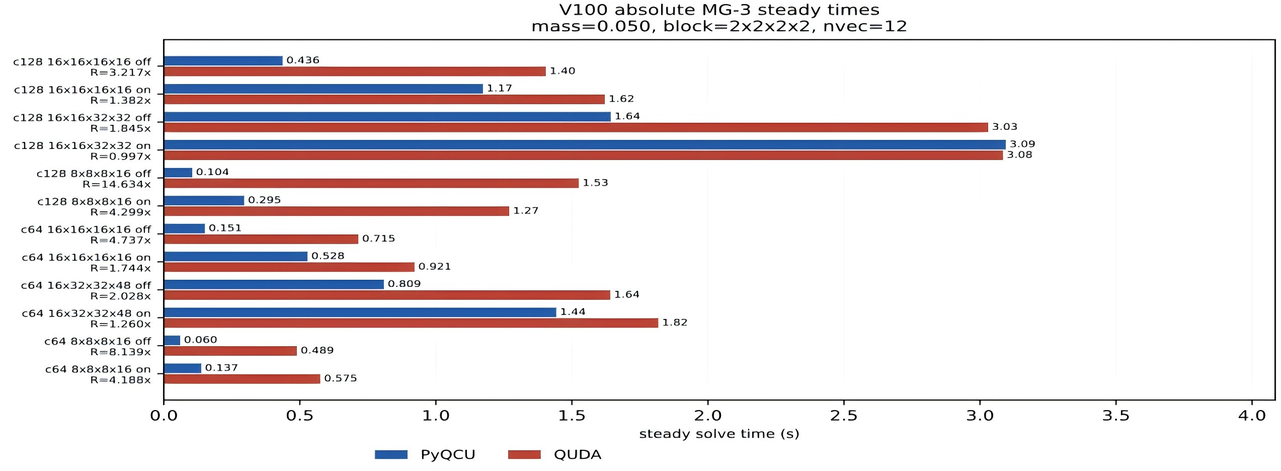


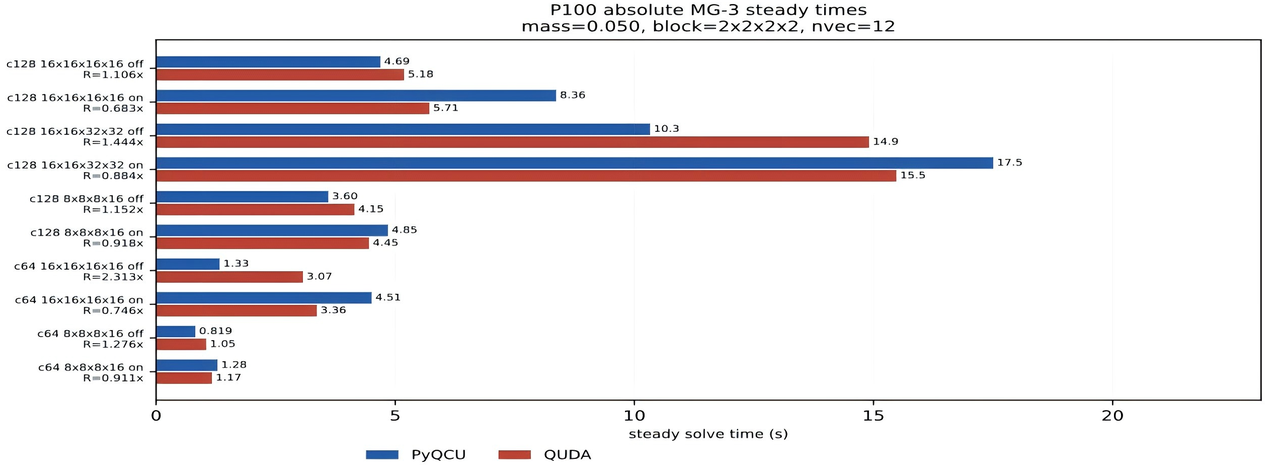


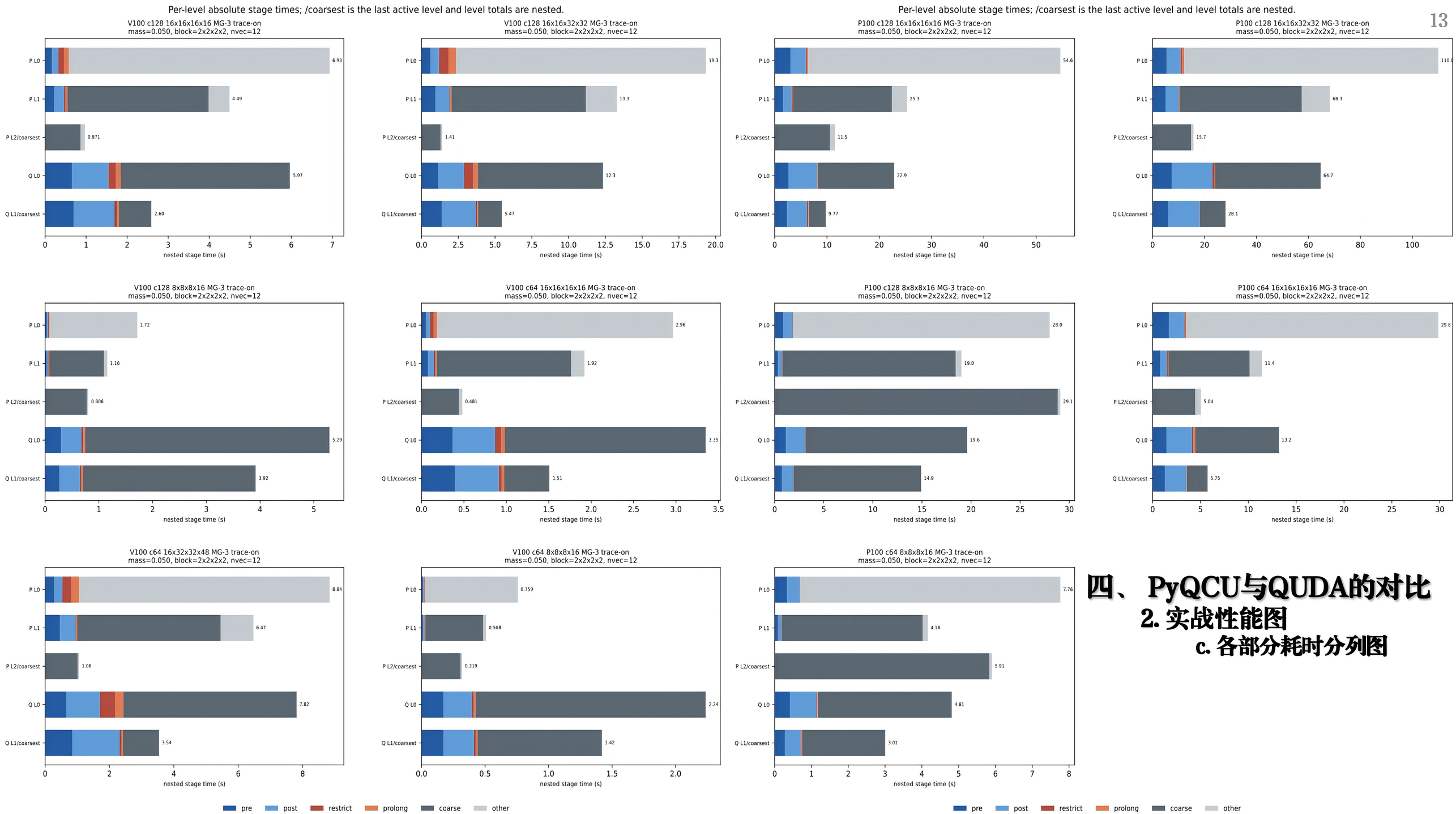
四、 PyQCU与QUDA的对比

2. 实战性能图

b. 各个测试组求解耗时分列图



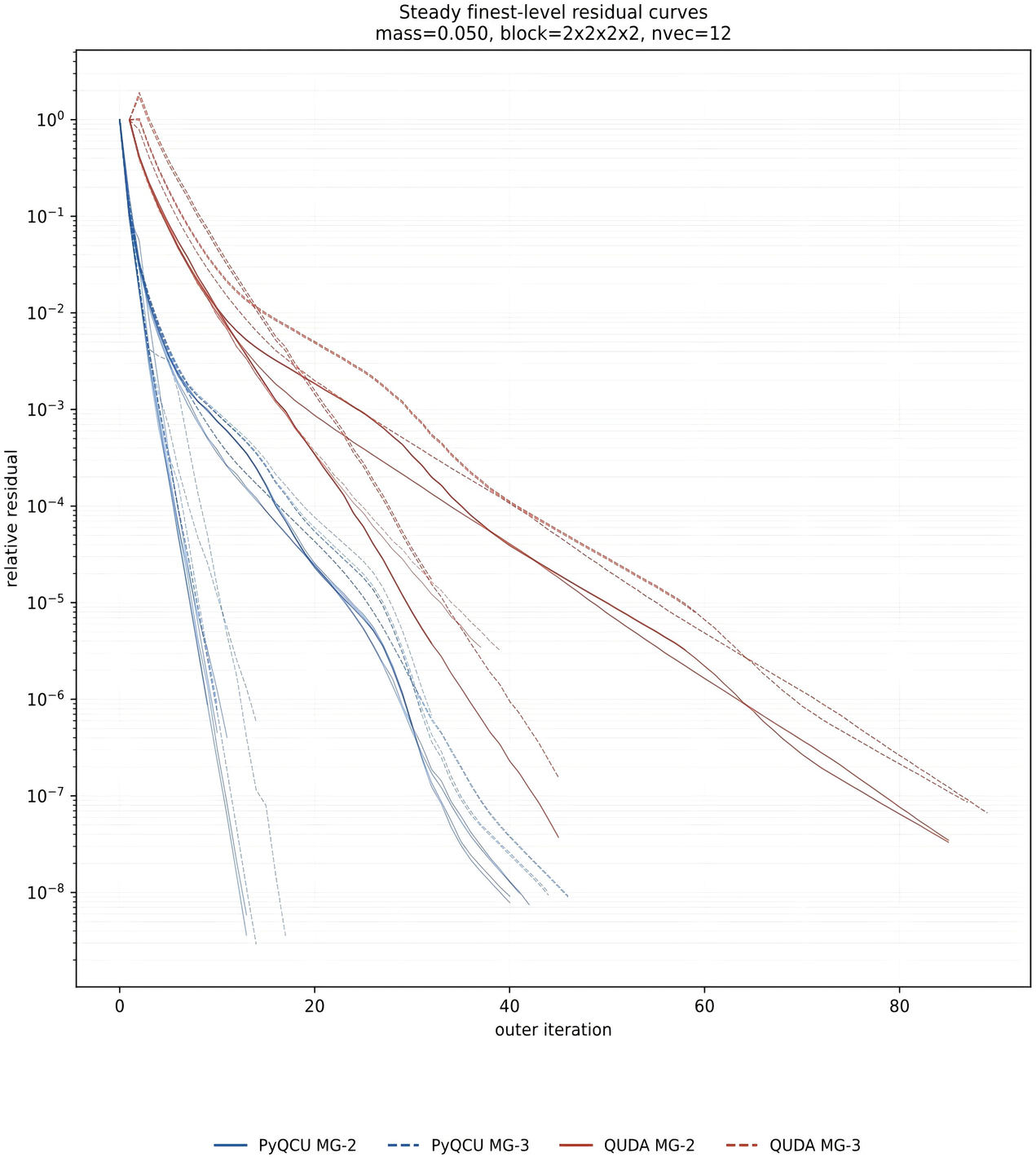


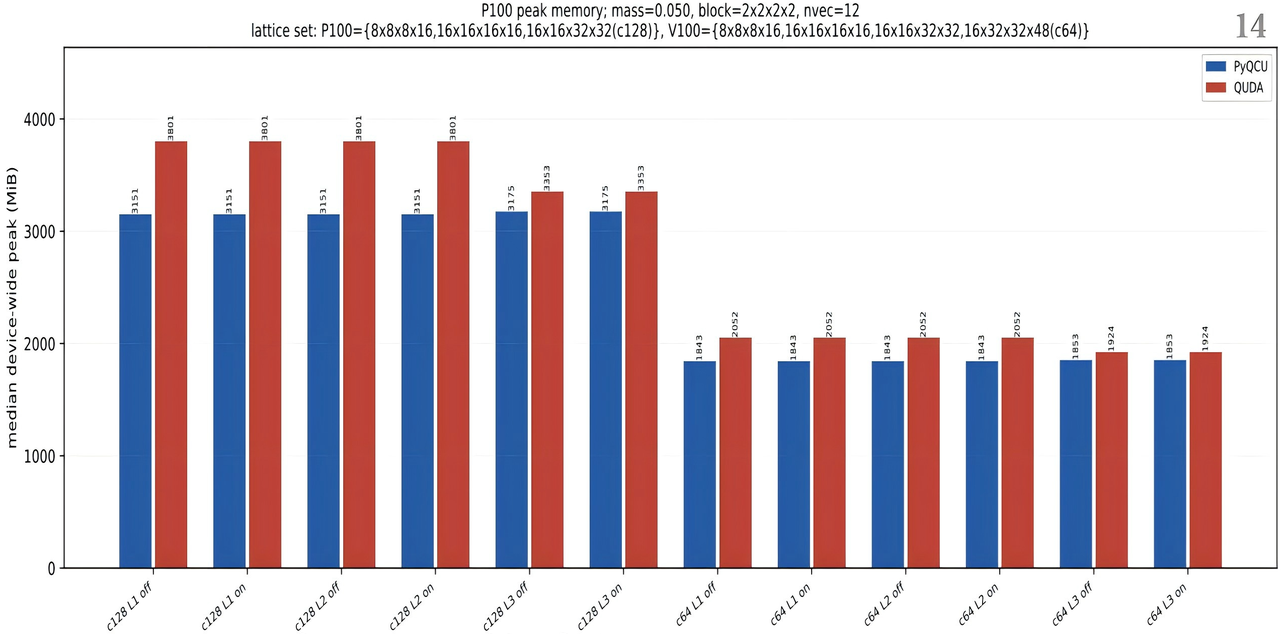


四、 PyQCU与QUDA的对比

2. 实战性能图

c. 各部分耗时分列图

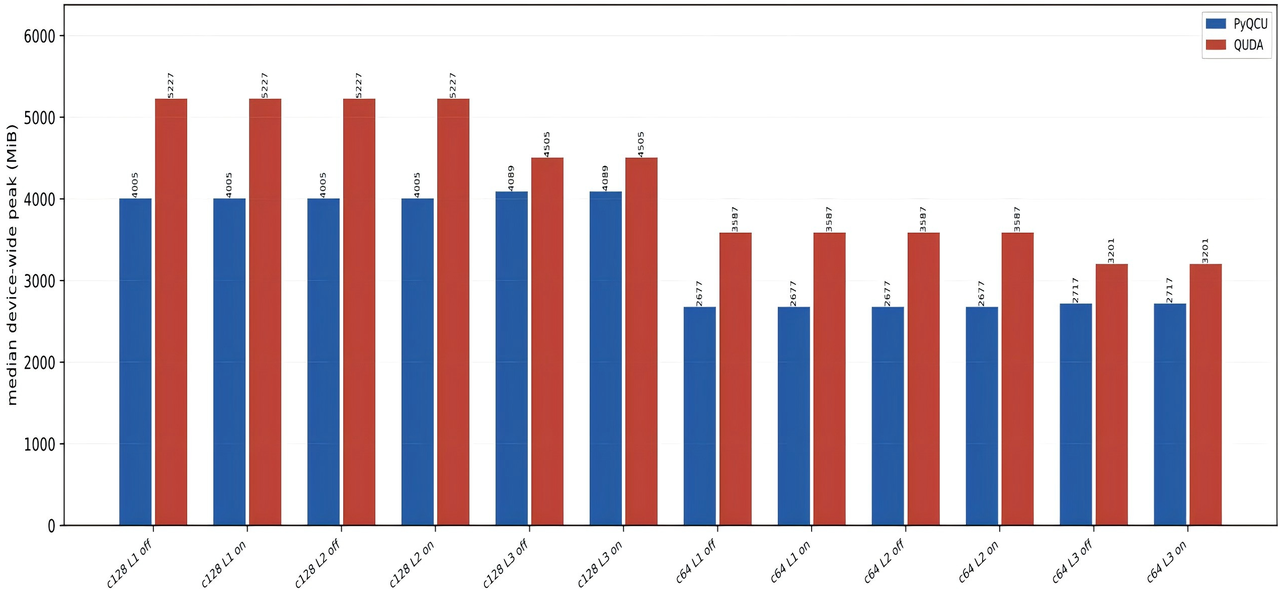




四、 PyQCU与QUDA的对比

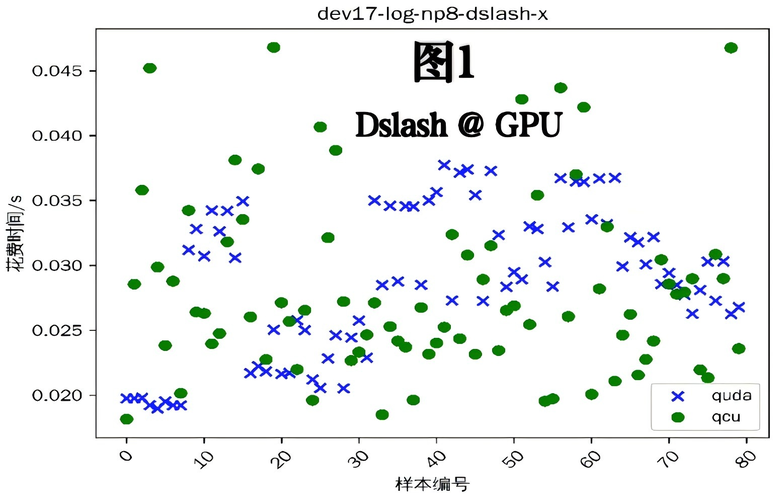
2. 实战性能图

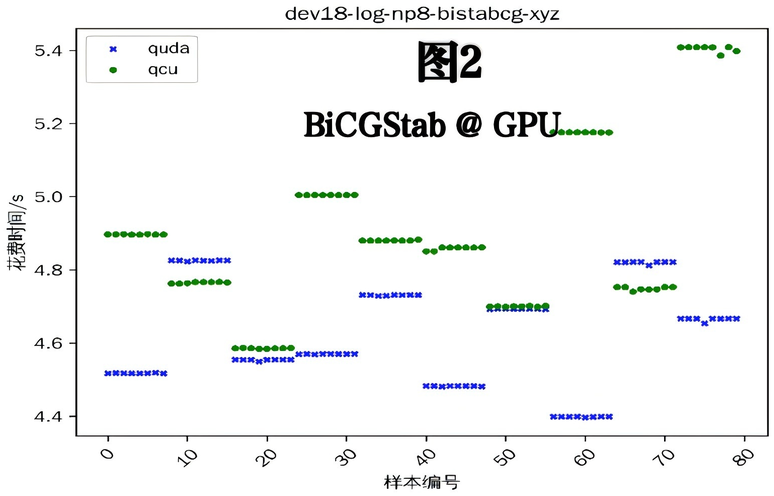
d. 求解时迭代收敛与运行时显存占用峰值一览图

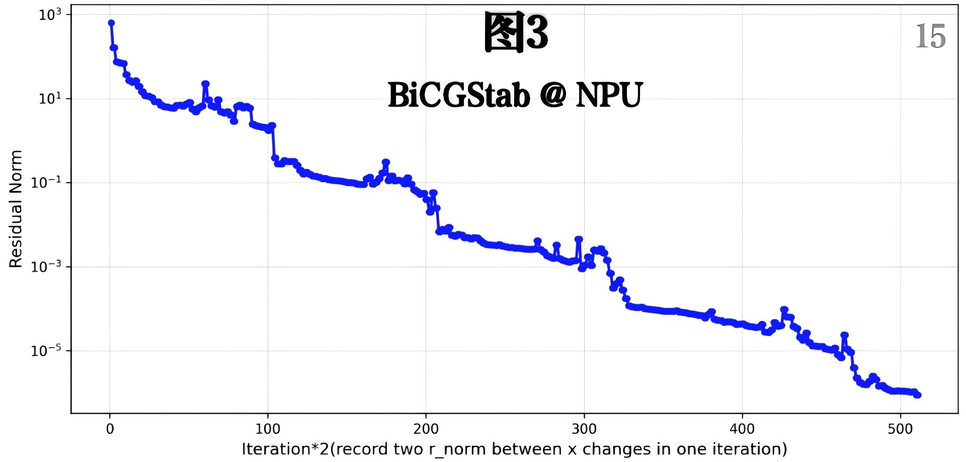


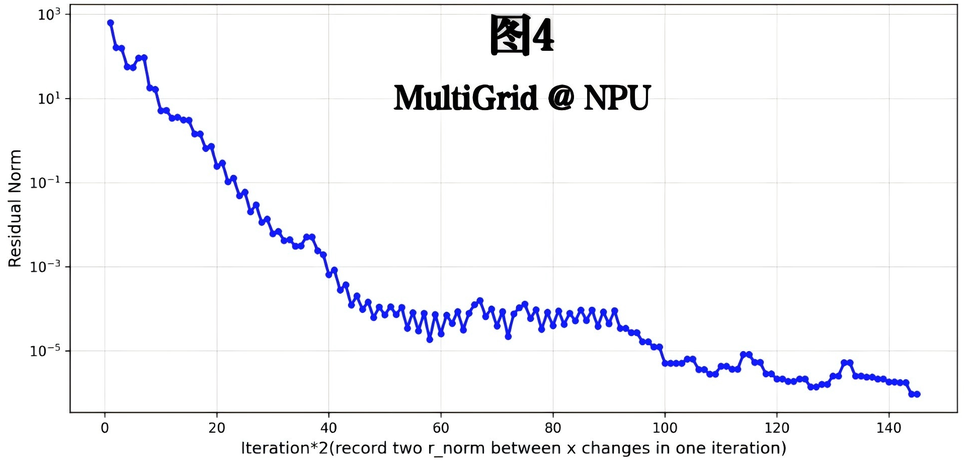
五、 PyQCU在国产计算平台

- 图一为单节点8卡NV-A100-SXM-80GB下QCU与QUDA的Wilson-Dslash的耗时图，图二为Wilson-BiCGStab的耗时图，此时QCU达到QUDA**约80%的性能**(单一精度)
- 图三为单节点8卡HW-ASCEND-910C-64GB下QCU的Clover-BiCGStab的残差图，图四为Clover-MultiGrid的残差图，此时后者**迭代数减少到约1/3**(使用TileLang优化)
- 表一为多节点128-1024卡DCU-BW1000-64GB下QCU求解器的强扩展性分析表，可见QCU在跨节点时仍具有**较强的可扩展性**









PyQCU：Mass为0.05，tol(x_o)为1e-12，数据类型为complex128，_BLOCK_SIZE_=256，使用节点绑定脚本，使用Wilson与Clover算子时，用面源求解传播子的DCU性能测试，其中（X:128，Y:128，Z:128.T:256）（具体数据参考https://gitee.com/zhangxin8069/PyQCU/tree/test9）

表1

序号	线程数	切分方式	等效单卡格点数量	WilsonBistabCg迭代数	WilsonBistabCg最终残差范数的平方(x_e)	WilsonBistabCg耗时/s	WilsonBistabCg单次迭代耗时/ms	WilsonBistabCg加速比	CloverBistabCg迭代数	CloverBistabCg最终残差范数的平方(x_e)	CloverBistabCg耗时/s	CloverBistabCg单次迭代耗时/ms	CloverBistabCg加速比
1	128	2，4，4，4	4194304	162	4.36466E-23	9.078154613	56.03799144	1.000	170	6.14E-23	28.5984921	168.2264241	1.000
2	256	4，4，4，4	2097152	156	9.40601E-23	6.953258005	44.5721667	1.257	159	6.36E-23	16.31604647	102.6166444	1.639
3	512	4，4，4，8	1048576	157	2.17205E-23	5.462116936	34.79055373	1.611	157	5.33213E-23	9.582966548	61.03800349	2.756
4	1024	4，4，8，8	524288	157	1.14308E-23	3.979191492	25.34516874	2.211	149	3.79539E-23	5.88710933	39.51080087	4.258

显存峰值占用大概为64GB*64卡*50%*2=4096GB

六、 总结与展望

总结

- SNSC[NVIDIA_A100-80GB*8](2024年11月):
  - 基础算子与求解器的串行与并行实现(Wilson-Dslash Clover-Dslash Wilson-CG Wilson-BiCGStab)
  - 单节点测试串行与并行性能符合预期
- 508[SUGON_DCU-64GB*1024](2025年8月):
  - 基础求解器的进一步串行与并行实现(Clover-CG Clover-BiCGStab)
  - 跨节点测试串行与并行性能符合预期，强拓展性与弱拓展性良好，对国产计算平台的初步适配成功
- ZHIYUAN[ASCEND_NPU-64GB*8](2026年4月):
  - MultiGrid求解器算法的初步验证(类quantum-mg方案)
  - 算法验证成功，开始针对非类CUDA生态进行适配(TileLang尝试)
- X99[NVIDIA_V100-32GB*1+NVIDIA_P100-16GB*2](2026年9月):
  - MultiGrid求解器的串行与并行实现(Wilson-MultiGrid Clover-MultiGrid)(类quda方案)
  - 单节点测试串行与并行性能符合预期，具备一定生产力价值，较大格子与并行实现以及MultiGrid求解器的最粗层子求解器存在较大优化余量

展望

目前本项目取得了一些阶段性进展，但是相对而言仍较为早期，核心算子与求解器仍有一定进步空间，日后要向国产计算平台做进一步的针对性适配与优化并尝试自主解决一些新的计算热点，最后希望在不久的将来，我们能够使用中国自己的算力与软件，即使用中国方案做出国际先进的格点工作!

谢谢!