ARM处理器上的格点QCD计算与优化

孙玮，毕玉江，程耀东

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.230200159

ARM 处理器上的格点 QCD 计算与优化

孙 玮[1] 毕玉江[1,2] 程耀东[1,2,3]
1 中国科学院高能物理研究所 北京 100049
2 四川天府新区宇宙线研究中心 成都 610213
3 中国科学院大学核科学与技术学院 北京 100049

摘 要 格点量子色动力学(格点 QCD)是高能物理领域中需要大规模并行计算的最主要应用之一,相关研究通常需要消耗大量计算资源,核心是求解大规模稀疏线性方程组。文中基于国产鲲鹏 920 ARM 处理器,研究了格点 QCD 的计算热点 Dslash,并将其扩展到 64 个节点(6144 核),展示了格点 QCD 计算的线性扩展性。基于 roofline 性能分析模型,发现格点 QCD 是典型的内存限制应用,并通过将 Dslash 中的 3×3 复幺正矩阵根据对称性压缩,将其性能提升约 22%。对于大规模稀疏线性方程的求解,在 ARM 处理器上探索了常用的 Krylov 子空间迭代算法 BiCGStab,以及近年来发展起来的前沿的 multigrid 算法,发现即使考虑预处理时间,在实际物理计算中使用 multigrid 算法相比 BiCGStab 依然有几倍至一个数量级的加速。此外,还考虑了鲲鹏 920 处理器上的 NEON 向量化指令,发现将其用于 multigrid 计算时可以带来约 20%的加速。因此,在 ARM 处理器上使用 multigrid 算法能极大地加速实际的物理研究。

关键词:格点 QCD;ARM 架构;多重网格算法;鲲鹏 920;NEON 向量化
中图法分类号 TP391

Lattice QCD Calculation and Optimization on ARM Processors

SUN Wei[1], BI Yujiang[1,2] and CHENG Yaodong[1,2,3]
1 Institute of High Energy Physics, Chinese Academy of Sciences, Beijing 100049, China
2 Tianfu Cosmic Ray Research Center, Chengdu 610213, China
3 School of Nuclear Science and Technology, University of Chinese Academy of Sciences, Beijing 100049, China

Abstract Lattice quantum chromodynamics (lattice QCD) is one of the most important applications of large-scale parallel computing in high energy physics, researches in this field usually consume a large amount of computing resources, and its core is to solve the large scale sparse linear equations. Based on the domestic Kunpeng 920 ARM processor, this paper studies the hot spot of lattice QCD calculation, the Dslash, which is applied on up to 64 nodes (6144 cores) and show the linear scalability. Based on the roofline performance analysis model, we find that lattice QCD is a typical memory bound application, and by using the compression of 3×3 complex unitary matrices in Dslash based on symmetry, we can improve the performance of Dslash by 22%. For the solving of large scale sparse linear equations, we also explore the usual Krylov subspace iterative algorithm such as BiCGStab and the newly developed state-of-art multigrid algorithm on the same ARM processor, and find that in the practical physics calculation the multigrid algorithm is several times to a magnitude faster than BiCGStab, even including the multigrid setup time. Moreover, we consider the NEON vectorization instructions on Kunpeng 920, and there is up to 20% improvement for multigrid algorithm. Therefore, the use of multigrid algorithm on ARM processors can speed up the physics research tremendously.

Keywords Lattice QCD, ARM architecture, Multigrid algorithm, Kunpeng 920, NEON vectorization

1 引言

粒子物理标准模型(Standard Model)是高能物理研究的理论基础,也是人类迄今为止提出的最成功的科学理论之一,其中描述基本粒子夸克和胶子间强相互作用的理论被称为量子色动力学(Quantum Chromodynamics,QCD)。强相互作用将夸克结合成质子、中子,再进一步形成原子核,因此,对其进行精确计算对于理解宇宙的基本物质构成具有重要意义。

到稿日期:2023-02-22 返修日期:2023-04-21
基金项目:国家自然科学基金(12205311,11935017,12075253,12175063);中国博士后科学基金面上资助(2022M713154);高能物理研究所科技创新项目(E15451U2)
This work was supported by the National Natural Science Foundation of China (12205311, 11935017, 12075253, 12175063), China Postdoctoral Science Foundation (2022M713154) and Science and Technology Innovation Project of Institute of High Energy Physics (E15451U2).
通信作者:孙玮(sunwei@ihep.ac.cn)

一方面，由于强相互作用的渐近自由特性，在高能时可以通过微扰展开的费曼图方法来进行理论计算；另一方面，在低能时夸克胶子禁闭在强子内部，传统的费曼图方法不再适用。1974 年，Wilson 提出了非微扰的格点 QCD(lattice QCD)理论来研究色禁闭[1]，之后 Creutz 于 1980 年引入 Monte Carlo 方法[2]，使得格点 QCD 成为目前唯一的、从第一性原理出发并能持续改进计算误差、利用计算机直接研究强相互作用的理论。

格点 QCD 易于并行的计算模式和对计算资源的大量需求，使其成为高能物理领域对计算机时需求最大的研究方向之一，同时也成为了高性能计算领域的重要应用之一[3]。格点 QCD 相关的计算分别于 1995 年、1998 年、2006 年获得戈登贝尔奖。计算机软硬件的发展直接决定了格点 QCD 相关的物理研究的计算精度，使得我们可以在更加强大的高性能计算机上模拟更接近真实物理世界的理论。近十年来，随着异构计算的兴起，格点 QCD 也从传统的多核 CPU 计算扩展到包括 GPU 等加速卡的大规模并行计算模式[4-5]，研究内容包括在各种架构的超算与集群上开发和优化算法与软件。而近年来，ARM 架构处理器从移动端、PC 端逐渐进入高性能计算领域[6]，甚至目前世界超算 TOP500 榜单排名第二的日本的富岳，也是基于 ARM 架构的 A64FX 处理器。因此，将格点 QCD 计算从传统 x86 和 PowerPC 等架构的通用 CPU、Nvidia 和 AMD 的 GPU，以及专门定制的用于格点计算的处理器[7]扩展到 ARM 架构，并优化实现数值算法，对高效利用计算资源进行理论物理研究，以及拓展现有 ARM 处理器的应用生态都有重要意义。

目前针对 ARM 处理器的格点 QCD 研究很少，文献[8]介绍了 ARM SVE (Scalable Vector Extension)向量化指令集在 Grid 软件中的应用，文献[9]则在 Bridge＋＋软件中探索了日本富岳超算 A64FX 处理器上的 512 位 SVE 指令集。本文基于支持 128 位 NEON 向量化指令的华为鲲鹏 920 ARM 处理器[10]，研究格点 QCD 的计算热点 Dslash，并探索多重网格算法等前沿的稀疏线性方程求解算法，加速物理计算，探索 ARM 处理器在格点 QCD 这一基础物理前沿计算领域的应用。

2　格点 QCD 计算热点

2.1　Stencil 运算

格点 QCD 将连续时空离散化为四维格点，进而可以通过 Monte Carlo 方法来数值计算路径积分。通常使用的离散化方案称为 Wilson-Dirac 形式，计算热点为求解大规模稀疏线性方程 $\boldsymbol{Mx}=\boldsymbol{b}$，其中 $\boldsymbol{M}=m+4-1/2\boldsymbol{D}$ 为 Wilson-Dirac 矩阵，m 为夸克质量参数，$\boldsymbol{D}$ 通常称为 Dslash 运算，定义为：

$$D_{x,y}=\sum_{\mu=1}^{4}\boldsymbol{U}_{\mu}(x)(1-\gamma_{\mu})\delta_{x+\hat{\mu},y}+\boldsymbol{U}_{\mu}(x-\hat{\mu})^{+}(1+\gamma_{\mu})\delta_{x-\hat{\mu},y} \tag{1}$$

其中 x,y 为时空格点坐标；μ 为时空 x,y,z,t 方向；$\boldsymbol{U}_{\mu}(x)$ 为 3×3 的复幺正矩阵(行列指标为色分量，在上式中略去)；$\delta_{i,j}$ 为 Kronecker delta 函数；$\boldsymbol{\gamma}_{\mu}$ 为四维时空的 Dirac γ 矩阵(行列指标为旋量分量，上式中略去)，具体为 4 个 4×4 的稀疏矩阵。

$$\boldsymbol{\gamma}_1=\begin{pmatrix}0&0&0&i\\0&0&i&0\\0&-i&0&0\\-i&0&0&0\end{pmatrix},\boldsymbol{\gamma}_2=\begin{pmatrix}0&0&0&-1\\0&0&1&0\\0&1&0&0\\-1&0&0&0\end{pmatrix},$$
$$\boldsymbol{\gamma}_3=\begin{pmatrix}0&0&i&0\\0&0&0&-i\\-i&0&0&0\\0&i&0&0\end{pmatrix},\boldsymbol{\gamma}_4=\begin{pmatrix}0&0&1&0\\0&0&0&1\\1&0&0&0\\0&1&0&0\end{pmatrix} \tag{2}$$

从式(1)可以看出，Wilson-Dirac 矩阵 $\boldsymbol{M}$ 在时空坐标上是近邻计算，具体为四维九点 Stencil 操作。对一个典型的格点研究，矩阵 $\boldsymbol{M}$ 的维数一般在 $O(10^9)$ 量级，因此对计算资源需求很大，加上 Stencil 运算的特点，使得格点 QCD 特别适合通过高性能并行计算来研究。图 1 给出了二维平面上的 Stencil 运算，作为四维时空中一个二维平面的图示。

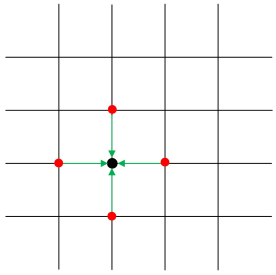

图 1　四维九点 Stencil 运算的图示(二维示意图)

Fig. 1　Illustration of four dimensional nine points Stencil operation (two dimensional schematic diagram)

2.2　计算环境

硬件方面，本文使用了国产的华为鲲鹏 920 ARM 处理器[10]，制程为 7nm，主频 2.6 GHz，支持 ARM v8.2 指令集。在服务器配置方面，单节点包含两颗 48 核 CPU，总计每节点 96 核。其中一级指令(L1i)和数据(L1d)缓存分别为 64 kB，二级缓存(L2)为 512 kB，三级缓存(L3)为 48 MB，单节点总计有 4 个 NUMA(Non-uniform Memory Access)节点，内存为 256 GB。

软件方面，我们基于著名的开源格点 QCD 计算框架 chroma[11]，实现和优化了本研究中的算法[1)]。在运行环境方面，使用了 GCC 编译套件和 OpenMPI 库，节点间的连接使用 RoCE(RDMA over Converged Ethernet)网络协议，具体信息如表 1 所列。

表 1　软件运行环境

Table 1　Software operation environments

	software	version
Compiler	GCC[12]	7.3.1
MPI	OpenMPI[13]	4.1.0
Network	RoCE	ConnectX-6
OS	CentOS	7.6

3　Dslash 性能分析

3.1　Dslash 运算的扩展性

Dslash 运算如式(1)所示，是近邻的 Stencil 运算，在并行

1) https://github.com/IHEP-LQCD/chromaxx, https://github.com/IHEP-LQCD/mg_proto

实现时，每个 MPI 进程计算一个子时空格点，相邻子格点的边界需要进行通信。为评估 Dslash 计算的扩展性，我们在每个节点上固定时空格点大小为 $N_s^3 \times N_t = 24^3 \times 48$（$N_s$ 和 N_t 分别为空间和时间方向的格点数），且每节点使用 96 个 MPI 进程，即每个 CPU 核心上一个 MPI 进程，通过 1，2，4，8，16，32，64 节点扩展至时空格点 $48^3 \times 384$，测试了 Dslash 随物理问题规模和计算规模变化的扩展性（编译器优化选项均为“-O3”）。对于 3×3 的复幺正矩阵 $\boldsymbol{U}_\mu(x)$，一般可用 18 个实数来表示，因此将此时的性能扩展结果用 C_{18} 标记，图 2 中用对数坐标展示了 Dslash 计算的可扩展性（其中 C_{12} 的定义见后文 3.2 节）。可以看到，在 ARM 服务器上，Dslash 在 64 节点（即 64×96=6144 核）内都具有很好的线性扩展性。

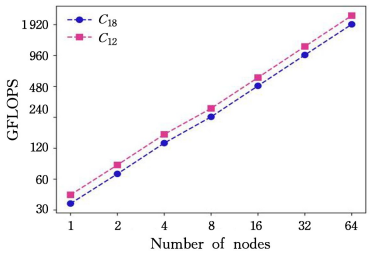

图 2 格点 QCD 计算热点 Dslash 的扩展性（对数坐标）

Fig. 2 Scalability of lattice QCD computing hot spot Dslash (log scale)

3.2 3×3 复矩阵的压缩

一般地，3×3 复矩阵 $\boldsymbol{U}$ 由 18 个实数表示，但在格点 QCD 中，由于非阿贝尔规范对称性，$\boldsymbol{U}$ 属于 $SU(3)$ 群，是行列式为 1 的复幺正矩阵，即满足 $\boldsymbol{U}^+\boldsymbol{U}=1$。因此，矩阵 $\boldsymbol{U}$ 的行列是非独立的，可以用更少的参数来表示。例如，将 $\boldsymbol{U}$ 写为：

$$\boldsymbol{U}=\begin{pmatrix}\vec{a}\\ \vec{b}\\ \vec{c}\end{pmatrix}=\begin{pmatrix}a_1 & a_2 & a_3\\ b_1 & b_2 & b_3\\ c_1 & c_2 & c_3\end{pmatrix} \tag{3}$$

则有 $\vec{c}=(\vec{a}\times\vec{b})^*$，即矩阵 $\boldsymbol{U}$ 的第三行（列）可以通过前两行（列）计算出来，也即 $c_i=(\boldsymbol{\varepsilon}_{ijk}a_jb_k)^*$，$\boldsymbol{\varepsilon}_{ijk}$ 为三阶全反对称张量。具体地，有：

$$c_1=(a_2b_3-a_3b_2)^*$$

$$c_2=(a_3b_1-a_1b_3)^*$$

$$c_3=(a_1b_2-a_2b_1)^*$$

因此，也可以只使用矩阵 $\boldsymbol{U}$ 中的前两行，通过浮点运算换取内存读写的方式，在计算时重建出第三行，达到压缩数据读写的目的。此时矩阵 $\boldsymbol{U}$ 可以通过 12 个实数来表示（记为 C_{12}），相应的 Dslash 扩展性能如图 2 所示。可以看到，Dslash 在 64 节点内依然呈现出很好的线性行为，C_{12} 相比 C_{18} 有约 22%的性能提升。

3.3 性能分析

为了从理论上理解 C_{12} 相比 C_{18} 的性能提升，本文考虑一个简单的 roofline 性能分析模型[14]，忽略所使用的 ARM 处理器的复杂多级缓存影响。对于式（1）中定义的 Dslash 运算，我们需要分析其算术强度（Arithmetic Intensity，AI）。由于 $\boldsymbol{\gamma}_\mu$ 矩阵是稀疏的，当作用于 3×4=12（3 为色自由度，4 为旋量自由度）个分量的复向量 $\boldsymbol{x}$ 时，不同旋量指标的结果之间并不独立，例如对于 $\boldsymbol{\gamma}_1$，有

$$(1-\boldsymbol{\gamma}_1)\boldsymbol{x}=\begin{pmatrix}1 & 0 & 0 & -i\\ 0 & 1 & -i & 0\\ 0 & i & 1 & 0\\ i & 0 & 0 & 1\end{pmatrix}\begin{pmatrix}\boldsymbol{x}_0\\ \boldsymbol{x}_1\\ \boldsymbol{x}_2\\ \boldsymbol{x}_3\end{pmatrix}$$

$$=\begin{pmatrix}\boldsymbol{x}_0-i\boldsymbol{x}_3\\ \boldsymbol{x}_1-i\boldsymbol{x}_2\\ i\boldsymbol{x}_1+\boldsymbol{x}_2\\ i\boldsymbol{x}_0+\boldsymbol{x}_3\end{pmatrix}=\begin{pmatrix}\boldsymbol{y}_0\\ \boldsymbol{y}_1\\ i\boldsymbol{y}_1\\ i\boldsymbol{y}_0\end{pmatrix} \tag{4}$$

其中，$\boldsymbol{x}_0,\cdots,\boldsymbol{x}_3,\boldsymbol{y}_0,\boldsymbol{y}_1$ 均为包含 3 个色自由度的复向量。可以看出，由于 $\boldsymbol{\gamma}$ 矩阵的稀疏性和对称性，$(1-\boldsymbol{\gamma}_\mu)\boldsymbol{x}$ 的乘积只需计算出上二分量 $\boldsymbol{y}_0$ 与 $\boldsymbol{y}_1$，而下二分量可以通过简单的复数操作得到。

在计算浮点数运算时，通常将一次加（减）法或乘法均记为一次浮点运算（FLOP），则对于两个复数相加，FLOP 为 2，对于两个复数相乘，FLOP 为 6。因此，对于 C_{18} 情形，在每个时空点按式（4）计算 $\boldsymbol{y}_0,\boldsymbol{y}_1$ 时，其每个色分量需要一次复数加（减）法，总计 2×3×2=12 FLOP，然后再进行 3×3 复矩阵 $\boldsymbol{U}$ 与向量 $\boldsymbol{y}_0,\boldsymbol{y}_1$ 的乘法，其中每一行的矩阵乘法包含 3 个复数乘法与两个复数加法，也即 3×6+2×2=22 FLOP，一个完整的矩阵乘法则需 3×22=66 FLOP，因此对于计算 $\boldsymbol{y}_0,\boldsymbol{y}_1$ 的矩阵向量乘法需要 2×66=132 FLOP。故 C_{18} 情形的每个格点上 Dslash 运算的浮点数总计为：

$$F_{18}=8\times12+8\times132+7\times12\times2=1320\,\text{FLOP}$$

其中，8 为四维 Dslash Stencil 运算的 8 个方向，7 来自式（1）中对 $\boldsymbol{\mu}$ 求和时的 7 次复向量（4×3=12 维）加法。再考虑 Dslash 运算的内存读写操作，在 C_{18} 情形，每个格点上的计算需要从内存中读入相邻的 8 个 3×3 复矩阵 $\boldsymbol{U}$ 及 4×3=12 维复向量，并将最后计算的结果，即一个 4×3=12 维复向量写入内存。因此，对于双精度计算，C_{18} 下每个格点上 Dslash 运算的内存读写量为：

$$B_{18}=(8\times18+8\times24+24)\times\text{sizeof}(double)=2880\,\text{Byte}$$

则 C_{18} 下的算术强度为：

$$I_{18}=\frac{F_{18}}{B_{18}}=\frac{1320}{2880}\approx0.458\,\text{FLOP/Byte}$$

然而，对于包括鲲鹏 920 在内的大多数现代处理器，其峰值浮点运算性能都要远高于内存带宽（鲲鹏 920 的双精度理论峰值约为 1 TFLOPS，内存带宽约 188 GB/s），因此，格点 QCD 的计算热点 Dslash 在这些处理器上都是内存限制的（Memory bound），而加速计算则可以探索增加计算热点算术强度的新算法。

另一方面，对于将矩阵 $\boldsymbol{U}$ 压缩成两行的 C_{12} 情形，在按式（3）重建第三行 $\vec{c}$ 时增加的浮点运算为 3×（2×6+2）=42 FLOP，而内存读写时只需读入复矩阵 $\boldsymbol{U}$ 的前两行，因此其内存读写量为（8×12+8×24+24）×sizeof（*double*）=2496 Byte，则 C_{12} 的算术强度为：

$$I_{12}=\frac{1320+4\times42}{2496}\approx0.596\,\text{FLOP/Byte}$$

因此从理论上的算术强度增加来看，C_{12} 相比 C_{18} 的加速应该约为 $\frac{I_{12}}{I_{18}}=1.30$，这与我们在图 3 中实际测得的性能

接近。以上分析虽然基于一个忽略了缓存等影响的简单 roofline 模型，但对于判断格点 QCD 的计算热点 Dslash 是内存限制很有必要。

4　稀疏线性方程求解

Dslash 计算是格点 QCD 中 Dirac 方程的基本组成部分，本节将基于 Dslash 计算研究求解 Dirac 方程的算法，即夸克传播子的求解，也即求解大规模稀疏线性方程 $\boldsymbol{Mx}=\boldsymbol{b}$ 时的算法。对于典型的格点计算，矩阵 $\boldsymbol{M}$ 的维数约为 $O(10^9)$，因此无法使用直接解法，而需采用 Krylov 子空间迭代算法，如常见的 CG[15]，BiCG[16] 等算法。由于常见的 Krylov 子空间迭代算法的收敛条件数(Condition Number)反比于矩阵 $\boldsymbol{M}$ 的最小本征值，因此当 Wilson-Dirac 矩阵 $\boldsymbol{M}$ 中的夸克质量参数 m 接近物理的夸克质量时，这些算法往往难以收敛，而且随着夸克质量的降低，迭代不收敛会变得越发显著，这称为临界降速(Critical Slowing Down)问题。本节我们将使用实际物理计算中通过 Monte Carlo 算法产生的复幺正矩阵 $\boldsymbol{U}$(每个时空点有 4 个 3×3 复幺正矩阵，所有时空点的集合称为一个规范场组态)，在$16^3\times128$ 的时空格点上，探索不同夸克质量参数下 $\boldsymbol{Mx}=\boldsymbol{b}$ 的求解。

4.1　BiCGStab 算法

BiCGStab (BiConjugate Gradient Stabilized)[17] 是常见的求解稀疏线性方程的单网格 Krylov 子空间迭代算法，具体算法如算法 1 所示，其计算核心是在 Krylov 子空间中迭代计算矩阵 $\boldsymbol{M}$ 与向量的乘法，以及 $\|r\|=\|\boldsymbol{Mx}-\boldsymbol{b}\|$，直到残差 $\|r\|$ 小于收敛精度后停止迭代。对于物理研究中的 $N_s^3\times N_t=16^3\times128$ 的规范场组态，我们在表 2 所列的一系列夸克质量参数 m 上应用无预处理的 BiCGStab 算法求解 $\boldsymbol{Mx}=\boldsymbol{b}$，在单个 ARM 计算节点上考察了求解时间(Time to Solution)，记为 $T_{\text{bcg}}^{\text{inv}}$，其中收敛精度为 $\varepsilon=10^{-8}$，结果如图 3 中蓝点(BICGSTAB)所示。由图 3 可知，当夸克质量参数 m 越靠近左侧时(即物理的夸克质量越轻时)，BiCGStab 算法的求解迅速变得难以收敛，即上文提到的临界降速问题。而这只是一次 $\boldsymbol{Mx}=\boldsymbol{b}$ 的求解，实际的物理计算需要在相同的规范场组态 $\boldsymbol{U}$，即相同的矩阵 $\boldsymbol{M}$ 上求解大量不同源向量 $\boldsymbol{b}$ 下的解 $\boldsymbol{x}$，且通常需要求解$4\times3\times N_t$次，其中 4 为旋量自由度数，3 为色自由度。对于$16^3\times128$ 的格子，则需要求解 1 536 次，因此对于轻质量夸克，BiCGStab 等类似单网格算法的临界降速问题严重地影响了物理计算。

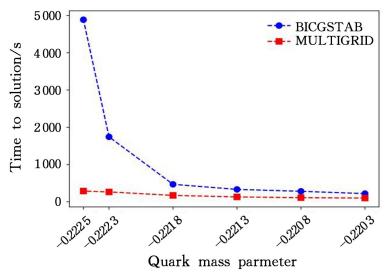

图 3　BiCGStab 与 multigrid 算法求解 $\boldsymbol{Mx}=\boldsymbol{b}$ 时的对比
(电子版为彩图)

Fig. 3　Comparison of BiCGStab and multigrid algorithm for solving $\boldsymbol{Mx}=\boldsymbol{b}$

求解 $\boldsymbol{Mx}=\boldsymbol{b}$ 的 BiCGStab 算法的伪代码如算法 1 所示。

算法 1　求解 $\boldsymbol{Mx}=\boldsymbol{b}$ 的 BiCGStab 算法

1. $r_0=\mathbf{b}-\mathbf{M}x_0$($x_0$ 为初始猜测解)
2. $p_0=r_0$
3. while $\|r_j\|>\varepsilon$ do (为收敛精度)
4. $\alpha_j=\dfrac{(r_j,r_0^*)}{(\mathbf{M}p_j,r_0^*)}$
5. $s_j=r_j-\alpha_j\mathbf{M}p_j$
6. $\omega_j=\dfrac{(\mathbf{M}s_j,s_j)}{(\mathbf{M}s_j,\mathbf{M}s_j)}$
7. $x_{j+1}=x_j+\alpha_j p_j+\omega_j s_j$
8. $r_{j+1}=s_j-\omega_j\mathbf{M}s_j$
9. $\beta_j=\dfrac{\alpha_j}{\omega_j}\times\dfrac{(r_{j+1},r_0^*)}{(r_j,r_0^*)}$
10. $p_{j+1}=r_{j+1}+\beta_j(p_j-\omega_j\mathbf{M}p_j)$
11. end while

4.2　Multigrid 算法

近年来，在偏微分方程数值计算中前沿的多重网格(multigrid)算法被引入了格点 QCD 计算中[18]，但研究多集中在 GPU 上的 multigrid 实现[19]，因此支持国产 ARM 处理器的 multigrid 实现就显得尤为重要。典型的 multigrid 算法包含如下预处理(setup)过程：

(1)计算 Wilson-Dirac 矩阵 $\boldsymbol{M}$ 的近零本征向量(near null vector)$\boldsymbol{v}_i$，$i=1,\cdots,n$，其满足 $\boldsymbol{M}\boldsymbol{v}_i\approx0$。

1)给定随机初始值x_0，应用 BiCGStab 等 Krylov 子空间算法迭代 k 次得到解向量$\boldsymbol{v}_i$。

2)重复上一步 n 次，得到 n 个解向量构成的近零本征向量集合v_i，$i=1,\cdots,n$。

(2)由v_i，$i=1,\cdots,n$ 构造 prolongation 算符 P 和 restriction 算符 R，通常取 $R=P^+$。

(3)由 R 和 P 通过 Galerkin 投影构造粗网格上的 Wilson-Dirac 矩阵 $\boldsymbol{M}_c=P^+MP$。

完成上述预处理后，就可以经过如图 4 所示的两层 V-Cycle 或更多层级的网格来迭代求解 $\boldsymbol{Mx}=\boldsymbol{b}$，将原始细网格上难以收敛的计算通过粗粒化在多重网格上完成。

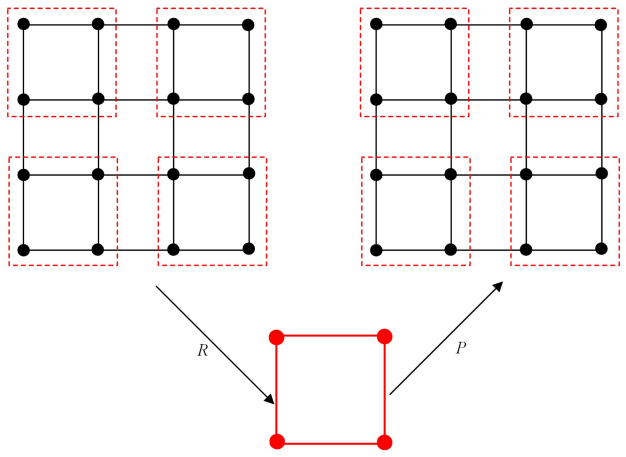

图 4　两层 multigrid 算法 V-Cycle 过程示意图

Fig. 4　Schematic diagram of two level V-Cycle process of multigrid algorithm

但格点 QCD 领域的 multigrid 算法较为复杂，一般是

针对特定硬件进行实现，如文献[19]在 NVIDIA GPU 上的实现。实际使用中，multigrid 算法一般是作为一个灵活的 Krylov 子空间迭代算法的预条件子（Preconditioner）来使用的。本研究选择 FGMRES（Flexible Generalized Minimum RESidual）[20]算法作为外部的 Krylov 迭代算法，将此时的算法称为 FGMRES-MG（下文中以 multigrid 代指 FGMRES-MG），具体如算法 2 所示。

算法 2 求解 $\boldsymbol{Mx}=\boldsymbol{b}$ 的 FGMRES-MG 算法

1. $r_0=\boldsymbol{b}-\boldsymbol{M}\boldsymbol{x}_0$（$\boldsymbol{x}_0$为初始猜测解）
2. $\beta=\|r_0\|$，$v_1=\frac{r_0}{\beta}$
3. for j=1,…,m do
4. $z_j=\widetilde{M}^{-1}v_j$（$\widetilde{M}^{-1}$表示应用 multigrid）
5. $w=M z_j$
6. for i=1,…,jdo
7. $h_{i,j}=(w,v_i)$
8. $w=w-h_{i,j}v_i$
9. end for
10. $h_{j+1,j}=\|w\|$，$v_{j+1}=\frac{w}{h_{j+1,j}}$
11. $Z_m:=[z_1,\cdots,z_m]$
12. $\bar{H}_m:=\{h_{i,j};1\leqslant i\leqslant j+1,1\leqslant j\leqslant m\}$
13. end for
14. $y_m=\underset{y}{\operatorname{argmin}}\|\beta e_1-\bar{H}_m y\|$，$x_m=x_0+Z_m y_m$
15. if convergence then
16. quit
17. else $x_0\leftarrow x_m$，go to 1
18. end if

与上文中 BiCGStab 算法的计算一致，我们用相同的夸克质量参数 m，在单个 ARM 节点上求解了 $16^3\times128$ 的相同规范场组态上的稀疏线性方程 $\boldsymbol{Mx}=\boldsymbol{b}$，收敛精度依然是 $\varepsilon=10^{-8}$，并应用了 3 层网格，即原始网格大小为 $N_s^3\times N_t=16^3\times128$，第二层网格为 $4^3\times32$，第三层为 $4^3\times16$，预处理过程中近零本征向量的个数分别为 24 和 32，具体求解时间如表 2 所列。其中 $T_{\text{mg}}^{\text{setup}}$ 为 multigrid 预处理时间，$T_{\text{mg}}^{\text{inv}}$ 为预处理后单次求解 $\boldsymbol{Mx}=\boldsymbol{b}$ 的计算时间。可以看到，在我们考察的夸克质量参数范围内，使用 multigrid 后相比 BiCGStab 的单次计算速度可加快数倍至一个数量级。更直观地，如图 3 中红点（MULTIGRID）所示，multigrid 算法在我们考虑的所有夸克质量参数内都可以很快收敛，反观 BiCGStab 则迅速变得难以收敛，因此 multigrid 算法在一定程度上解决了临界降速问题。

表 2 BiCGStab 与 Multigrid 算法在求解稀疏线性方程 $\boldsymbol{Mx}=\boldsymbol{b}$ 时的对比

Table 2 Comparison of BiCGStab and multigrid algorithm for solving spare linear equation $\boldsymbol{Mx}=\boldsymbol{b}$

quark mass parameter m	$T_{\text{bcg}}^{\text{inv}}$/s	$T_{\text{mg}}^{\text{setup}}$/s	$T_{\text{mg}}^{\text{inv}}$/s	S_1	S_2
−0.2203	220	3286	101	2.18	2.13
−0.2208	282	3285	111	2.54	2.49
−0.2213	333	3284	131	2.54	2.50
−0.2218	469	3290	172	2.73	2.69
−0.2223	1743	3286	265	6.58	6.52
−0.2225	4886	3270	288	16.97	16.84

值得注意的是，multigrid 算法的预处理本身会花费大量时间，如表 2 中的 $T_{\text{mg}}^{\text{setup}}$ 所列，但好在实际的物理计算需要在相同的矩阵 $\boldsymbol{M}$、不同的源向量 $\boldsymbol{b}$ 上求解很多次 $\boldsymbol{Mx}=\boldsymbol{b}$，而此时第一次计算时的预处理过程在后面计算时可以重用，因此我们定义 multigrid 相比 BiCGStab 算法的加速比为：

$$S_1=\frac{T_{\text{bcg}}^{\text{inv}}}{T_{\text{mg}}^{\text{inv}}}$$
$$S_2=\frac{T_{\text{bcg}}^{\text{inv}}\times12\times N_t}{T_{\text{mg}}^{\text{setup}}+T_{\text{mg}}^{\text{inv}}\times12\times N_t} \tag{5}$$

其中，S_1 为单次求解时忽略预处理时间的 multigrid 相比 BiCGStab 算法的加速，S_2 为包括 multigrid 预处理时间，在同一稀疏矩阵 $\boldsymbol{M}$ 上进行 $4\times3\times N_t$ 次求解的加速比。从表 2 可以看到，在实际的物理计算中，即使包括预处理时间，multigrid 相比 BiCGStab 依然有数倍至一个数量级的加速，这对加速物理研究有重要意义。

另一方面，现代处理器多支持单指令多数据（Single Instruction Multiple Data，SIMD）的向量化指令，如 x86 架构的 SSE，AVX，AVX512[21]等。相应地，ARM 也支持 SIMD 指令，如鲲鹏 920 上的 128 位 NEON[22]指令，因此我们也尝试将 multigrid 计算中部分耗时的矩阵运算，如粗粒化时 $\boldsymbol{M}_c=P^+MP$ 的构造通过手动使用 intrinsic function 实现 NEON 向量化。例如，对于双精度浮点数的加法，可以通过如下示例代码实现手动 NEON 向量化。

```
#include <arm_neon.h>
void add_neon(double * out,
              const double * input1,
              const double * input2)
{
    float64x2_t v1=vld1q_f64(input1);
    float64x2_t v2=vld1q_f64(input2);
    float64x2_t v0=vaddq_f64(v1,v2);
    vst1q_f64(out,v0);
}
```

其中 vld1q_f64 将两个 double 数据从内存中读入 128 位寄存器，vaddq_f64 对两个 128 位寄存器里的两组数据同时进行加法，最后 vst1q_f64 将结果寄存器中的数据写回内存。图 5 给出了手动NEON 向量化后的加速，可以看到，multigrid 算法在我们考察的夸克质量范围内有最高约 20% 的加速。一方面，手动 NEON 向量化带来的加速有限，但代码复杂性大幅提高，增加了后期的维护与纠错的困难程度。另一方面，ARM 处理器也有支持更高位宽的向量指令，如富岳超算的 SVE 向量指令集，将会为格点计算带来更高的加速，此时仔细考虑向量化是很有必要的。

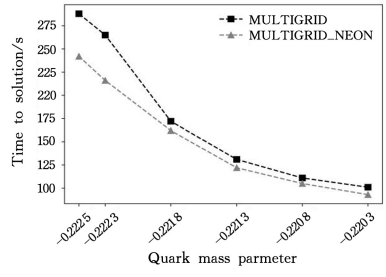

图 5 NEON 向量化的 multigrid 算法求解 $\boldsymbol{Mx}=\boldsymbol{b}$ 时的加速

Fig. 5 Speed up of multigrid algorithm with NEON vectorization in solving $\boldsymbol{Mx}=\boldsymbol{b}$

结束语　本文基于国产鲲鹏 920 ARM 处理器，研究了高能物理领域的典型高性能计算应用格点 QCD。通过将计算热点 Dslash 扩展到 64 个节点(6144 核)，展示了格点 QCD 计算很好的线性扩展性。基于 roofline 性能分析模型，从理论上理解了格点 QCD 是典型的内存限制的应用，为程序与算法优化奠定了理论基础，并通过将 Dslash 中的 3×3 复幺正矩阵根据对称性由前两行来表示，可以将性能提升约 22%，这与我们基于 roofline 模型的分析接近。

在实际物理计算中求解夸克传播子时需要大量的稀疏线性方程 $\boldsymbol{Mx}=\boldsymbol{b}$ 的求解，我们在 ARM 处理器上探索了常用的 Krylov 子空间迭代算法 BiCGStab，以及近年来发展起来的前沿的 multigrid 算法，发现即使考虑预处理时间，在实际物理计算中使用 multigrid 算法相比 BiCGStab 依然有几倍至一个数量级的加速。此外，还考虑了鲲鹏 920 处理器上的 NEON 向量化指令，发现将其用于 multigrid 中的部分计算时可以带来约 20%的加速。因此，在 ARM 处理器上使用 multigrid 算法能极大地加速实际的物理研究。

本文对基于格点 QCD 的强相互作用的研究有实际意义，目前基于 ARM 集群和这些算法的研究，已产生了一些有趣的物理成果，如文献[23-24]。此外，本研究也在一定程度上扩展了国产处理器与超算的应用生态。

参 考 文 献

[1] WILSON G K. Confinement of Quarks[J]. Physical Review D, 1974, 10(8): 2445-2459.

[2] CREUTZ M. Monte Carlo Study of Quantized SU(2) Gauge Theory[J]. Physical Review D, 1980, 21(8): 2308-2315.

[3] HABIB S, ROSER R, GERBER R, et al. ASCR/HEP Exascale Requirements Review Report[J]. arXiv: 1603.09303, 2016.

[4] EGRI G, FODOR Z, HOELBLING C, et al. Lattice QCD as a video game [J]. Computer Physics Communications, 2007, 177(8): 631-639.

[5] CLARK M, BABICH R, BARROS K, et al. Solving lattice QCD systems of equations using mixed precision solvers on GPUs[J]. Computer Physics Communications, 2010, 181(9): 1517-1528.

[6] JACKSON A, TURNER A, WEILAND M, et al. Evaluating the Arm Ecosystem for High Performance Computing[C]// Proceedings of the Platform for Advanced Scientific Computing Conference. 2019: 1-11.

[7] CHEN D, CHRIST N, DONG Z, et al. QCDOC: A 10-teraflops scale computer for lattice QCD[J]. Nuclear Physics B－Proceedings Supplements, 2001, 94(1): 825-832.

[8] MEYER N, PLEITER D, SOLBRIG S, et al. Lattice QCD on upcoming Arm architectures[C]// The 36th Annual International Symposium on Lattice Field Theory. 2018.

[9] ISHIKAWA K, KANAMORI I, MATSUFURU H. Multigrid solver on Fugaku[C]// The 39th Annual International Symposium on Lattice Field Theory. 2021.

[10] XIA J, CHENG C, ZHOUX, et al. Kunpeng 920: The First 7-nm Chiplet-Based 64-Core ARM SoC for Cloud Services[J]. IEEE Micro, 2021, 41(5): 67-75.

[11] EDWARDS R, JOO B. The Chroma software system for lattice QCD[J]. Nuclear Physics B－Proceedings Supplements, 2005, 140: 832-834.

[12] Free Software Foundation. GCC, the GNU Compiler Collection [EB/OL]. https://gcc.gnu.org.

[13] The Open MPI Project. Open MPI: Open Source High Performance Computing[EB/OL]. https://www.open-mpi.org.

[14] WILLIAMS S, WATERMAN A, PATTERSON D. Roofline: An Insightful Visual Performance Model for Multicore Architectures[J]. Communications of the ACM, 2009, 52(4): 65-76.

[15] MAGNUS H, EDUARD S. Methods of conjugate gradients for solving linear systems[J]. Journal of research of the National Bureau of Standards, 1952, 49: 409-435.

[16] WATSON A. Conjugate gradient methods for indefinite systems [M]// Numerical Analysis. Berlin: Springer, 1976: 73-89.

[17] VAN DER VORST H. Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems[J]. SIAM Journal on Scientific and Statistical Computing, 1992, 13(2): 631-644.

[18] BABICH R, BRANNICK J, BROWER R, et al. Adaptive Multigrid Algorithm for the Lattice Wilson-Dirac Operator[J]. Physical Review Letters, 2010, 105(20): 201602.

[19] CLARK M, JOO B, STRELCHENKO A, et al. Accelerating lattice QCD multigrid on GPUs using fine-grained parallelization [C]// Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2016: 1-12.

[20] YOUCEF S. A Flexible Inner-Outer Preconditioned GMRES Algorithm [J]. SIAM Journal on Scientific Computing, 1993, 14(2): 461-469.

[21] INTEL. Intel Instruction Set Extensions Technology[EB/OL]. https://www.intel.com/content/www/us/en/support/articles/000005779/processors.html.

[22] ARM. Arm NEON Technology [EB/OL]. https://developer.arm.com/Architectures/Neon.

[23] ZHANG R, SUN W, CHEN Y, et al. The glueball content of etac[J]. Physics Letters B, 2022, 827: 136960.

[24] ZHANG R, SUN W, CHENF, et al. Annihilation diagram contribution to charmonium masses[J]. Chinese Physics C, 2022, 46(4): 043102.

SUN Wei, born in 1992, Ph.D. His main research interests include high performance computing, lattice quantum chromodynamics and quantum computing.

(责任编辑：何杨)