物理学报 Acta Physica Sinica

Chinese Physical Society

Institute of Physics, CAS

格点量子色动力学蒸馏算法中关联函数的计算优化

张仁强 蒋翔宇 俞炯弛 曾充 宫明 徐顺

Calculation and optimization of correlation function in distillation method of lattice quantum chromodynamcis

Zhang Ren-Qiang Jiang Xiang-Yu Yu Jiong-Chi Zeng Chong Gong Ming Xu Shun

引用信息 Citation: Acta Physica Sinica, 70, 161201 (2021) DOI: 10.7498/aps.70.20210030

在线阅读 View online: https://doi.org/10.7498/aps.70.20210030

当期内容 View table of contents: http://wulixb.iphy.ac.cn

格点量子色动力学蒸馏算法中关联函数的计算优化*

张仁强[1)5)] 蒋翔宇[1)5)] 俞炯弛[2)] 曾充[3)] 宫明[1)5)] 徐顺[4)†]

1) (中国科学院高能物理研究所理论物理研究室, 北京 100049)

2) (浙江大学高分子科学与工程学系, 杭州 310058)

3) (浙江大学计算机科学与技术学院, 杭州 310058)

4) (中国科学院计算机网络信息中心, 北京 100190)

5) (中国科学院大学物理科学学院, 北京 100049)

(2021 年 1 月 6 日收到; 2021 年 3 月 26 日收到修改稿)

格点量子色动力学 (格点 QCD) 是一种以量子色动力学为基础, 被广泛应用于强相互作用相关计算的理论, 作为一种可以给出精确可靠理论结果的研究方法, 近年来随着计算机能力的提升, 正在发挥着越来越重要的作用. 蒸馏算法是格点 QCD 中计算强子关联函数的一种重要数值方法, 可以提高所计算物理量的信噪比. 但用它来构造关联函数时, 同样面临着数据量大和数据维数多的问题, 需要进一步提升计算效率. 本文开发了一套利用蒸馏算法产生夸克双线性算符的关联函数的程序, 利用 MPI (message passing interface, 消息传递接口, https://www.open-mpi.org), OpenMP (open multi-processing, 共享存储并行) 和 SIMD (single instruction multiple data, 单指令多数据流) 多级别优化技术解决其中计算性能瓶颈问题. 对程序进行了多方面的测试, 结果表明本文的设计方案能够支持大规模的计算, 在强扩展性测试下 512 个进程并行计算仍能达到 70% 左右的效率, 大大提升了计算关联函数的能力.

关键词: 格点量子色动力学, 蒸馏算法, 关联函数, 并行计算

PACS: 12.38.Gc **DOI:** 10.7498/aps.70.20210030

1 引 言

在自然界中有四种基本相互作用: 电磁相互作用、弱相互作用、强相互作用、引力相互作用, 它们决定了目前所知世界的物质运动规律. 比如摩擦力实际上是电磁相互作用的宏观效果; 而强相互作用决定了强子的基本性质, 如强子的质量[1]、衰变宽度[2]、形状因子[3]等, 同时将质子和中子结合在一起组成原子核的核力, 实际上是剩余强相互作用, 这种力决定了原子核的聚变和裂变以及质量等性质[4,5]. 因此研究强相互作用具有非常重要的意义. 描述强相互作用的理论是量子色动力学 (quantum chromodynamics, QCD)[6], 这种理论与电磁相互作用不同, 它在低能标是强耦合的, 这意味着无法用微扰展开的方式去近似计算强子的低能性质. 目前也发展出了一些研究低能 QCD 的方法, 如手征微扰论[7]、大 N_c 展开法[8]等, 这些方法都对理论做了一定程度的修改, 都依赖于模型. 而格点 QCD 则完全不需要任何的模型假设, 它直接从 QCD 出发, 利用蒙特卡罗方法进行可靠的数值模拟[9].

格点 QCD 计算应用需要借助大规模计算来实现组态的计算模拟, 需要巨大的高性能计算资源支撑, 其计算密集型和数据密集型特征对高性能计

* 国家重点研发计划 (批准号: 2017YFB0203203) 和国家自然科学基金 (批准号: 11775229, 11935017) 资助的课题.

† 通信作者. E-mail: xushun@sccas.cn

算技术的发展提出了不少挑战. 格点 QCD 是未来美国 E 级计算机的主要应用之一[10], 格点 QCD 在我国 E 级超算的应用还处于探索阶段, 有些算法方面的移植优化工作[11].

在格点 QCD 中, 通常需要计算如下形式的量[12]:

$$\langle O_{\text{observe}}\rangle = \frac{\int \mathcal{D}U O_{\text{observe}}[U]\mathrm{e}^{-S[U]}}{\int \mathcal{D}U \mathrm{e}^{-S[U]}}, \tag{1}$$

其中, $S[U]$表示 QCD 的作用量, $U(t,\boldsymbol{x})$是规范场, O_{observe}代表任意可观测量. 在欧几里得时空中, 可以将 (1) 式理解为简单的抽样模型

$$E(O_{\text{observe}}) = \frac{\int \mathrm{d}x\rho(x)O_{\text{observe}}(x)}{\int \mathrm{d}x\rho(x)}. \tag{2}$$

然后用蒙特卡罗方法去模拟, 将 $\rho(x)$理解为概率幅. 只要产生一组满足 $\mathrm{e}^{-S[U]}$分布的规范场 $U(t,\mathrm{x})$, 任意一个可观测量 O_{observe} 就是在这一组规范场上的统计平均值, 并且可以有效地估计其误差. 本工作研究的可观测量为夸克双线性算符的两点关联函数:

$$\begin{aligned} C_{\boldsymbol{p}}(t,t_0) &= \langle O^B(t,\boldsymbol{p})O^{A\dagger}(t_0,\boldsymbol{p})\rangle = \sum_{\boldsymbol{x},\boldsymbol{y}} \mathrm{e}^{\mathrm{i}\boldsymbol{p}(\boldsymbol{y}-\boldsymbol{x})}\langle \bar{\psi}(t,\boldsymbol{x})\Gamma^B\psi(t,\boldsymbol{x})\bar{\psi}(t_0,\boldsymbol{y})\Gamma^A\psi(t_0,\boldsymbol{y})\rangle \\ &= -\sum_{\boldsymbol{x},\boldsymbol{y}}\sum_{a,b,\alpha,\beta,\alpha',\beta'} \mathrm{e}^{\mathrm{i}\boldsymbol{p}(\boldsymbol{y}-\boldsymbol{x})}\langle \Gamma^B_{\alpha\beta}G^{a,b}_{\beta\alpha'}(t,\boldsymbol{x};t_0,\boldsymbol{y})\Gamma^A_{\alpha'\beta'}G^{b,a}_{\beta'\alpha}(t_0,\boldsymbol{y};t,\boldsymbol{x})\rangle \\ &\quad +\sum_{\boldsymbol{x},\boldsymbol{y}}\sum_{a,b,\alpha,\beta,\alpha',\beta'} \mathrm{e}^{\mathrm{i}\boldsymbol{p}(\boldsymbol{y}-\boldsymbol{x})}\langle \Gamma^B_{\alpha\beta}G^{a,a}_{\beta\alpha}(t,\boldsymbol{x};t,\boldsymbol{x})\Gamma^A_{\alpha'\beta'}G^{b,b}_{\beta'\alpha'}(t_0,\boldsymbol{y};t_0,\boldsymbol{y})\rangle, \end{aligned} \tag{3}$$

其中$\psi(t,\boldsymbol{x})$代表费米场, 形如矩阵$\boldsymbol{G}^{ab}_{\alpha\beta}$都表示传播子, 包含形如$\boldsymbol{G}(t,\boldsymbol{x};t_0,\boldsymbol{y})$的项被称为连通部分, 包含形如$\boldsymbol{G}(t,\boldsymbol{x};t,\boldsymbol{x})$的项被称为非连通部分. A 和 B 标记着不同的算符. 费米场和规范场写出所有指标的形式分别为$\psi^a_\alpha(t,\boldsymbol{x})$和$U^{ab}(t,\boldsymbol{x})$, 其中色指标$a,b\in\{0,1,2\}$, 旋量指标$\alpha,\beta\in 0,1,2,3$. 用$N_\mathrm{t}$表示时间维度上切片的个数, N_s表示空间维度中切片的个数, 那么就有$N_\mathrm{t}\times N_\mathrm{s}\times N_\mathrm{s}\times N_\mathrm{s}$个时空点. $\boldsymbol{G}^{ab}_{\alpha\beta}(t,\boldsymbol{x};t_0,\boldsymbol{y})$是包含完全信息的全传播子, 当$N_\mathrm{t}=128$、$N_\mathrm{s}=16$时, 它的维度为$128\times16^3\times4^2\times3^2$, 是一个巨大的矩阵, 计算时非常困难, 通常只计算固定t_0, $\boldsymbol{y}$时的值[13,14]. 在计算非连通部分时, 涉及一个从$(t,\boldsymbol{x})$到$(t,\boldsymbol{x})$的传播子, 不能像连通部分那样简单地通过把源固定某一时空点来减少计算量, 因此总体计算量非常大[15].

为了得到更好的统计精度, 近似地计算全传播子, 有人提出了蒸馏算法[16,17]. 这种算法在近似计算全传播子时需要的存储空间比传统方法小很多, 还可以很方便地进行算符构造, 即便在结束传播子的计算之后, 仍然可以改变涂摩 (smear)[18,19] 波函数, 提供了算符构造[20] 更多的自由空间. 蒸馏算法把传播子的计算转换到拉普拉斯算符的本征空间. 本征方程表示为$\sum_{\boldsymbol{y}}\Delta(t,\boldsymbol{x},\boldsymbol{y})V_c(t,\boldsymbol{y})=-\lambda_cV_c(t,\boldsymbol{x})$, 其中

$$\Delta(t,\boldsymbol{x},\boldsymbol{y}) \approx \sum_{c=0}^{N_\mathrm{v}} V_c(t,\boldsymbol{x})V_c^\dagger(t,\boldsymbol{y})f(\lambda_c), \tag{4}$$

N_v表示采用本征矢量的个数, N_v越大, 对拉普拉斯算符的近似越好, 也就越接近于计算全传播子. $f(\lambda_c)$是一个关于本征值的函数. 在蒸馏算法中只需要保存本征值、本征矢量和约化传播子 (perambulator):

$$\begin{aligned} \tau^{c_1c_2}_{\alpha\beta}(t,t_0) =& \sum_{\boldsymbol{x},\boldsymbol{y}}\sum_{ab}\boldsymbol{V}^{a\dagger}_{c_1}(t,\boldsymbol{x})\boldsymbol{G}^{ab}_{\alpha,\beta}(t,\boldsymbol{x};t_0,\boldsymbol{y}) \\ &\times \boldsymbol{V}^b_{c_2}(t_0,\boldsymbol{y}). \end{aligned} \tag{5}$$

约化传播子对应传统方法中的传播子$\boldsymbol{G}^{ab}_{\alpha,\beta}(t,\boldsymbol{x};t_0,\boldsymbol{y})$, 但是它具有很少的维度$(N_\mathrm{t}\times N_\mathrm{v}\times4)^2$, 比传统传播子更容易存储. 而本征矢量是一个列向量, 具有维度$N_\mathrm{t}\times N_\mathrm{s}^3\times N_\mathrm{v}\times3$, 也是一个体积比较小的量. 约化传播子的维度与传统传播子的维度的比值为$\dfrac{(N_\mathrm{t}\times N_\mathrm{v}\times4)^2}{(N_\mathrm{t}\times N_\mathrm{s}^3\times3\times4)^2}=\left(\dfrac{N_\mathrm{v}}{N_\mathrm{s}^3\times3}\right)^2$. 如在$N_\mathrm{t}=64$, $N_\mathrm{s}=32$的格子上, 为了得到一个较好的结果通常会令$N_\mathrm{v}\approx100$, 那么约化传播子的大小只是全传播子的$\dfrac{1}{983.04^2}$, 极大地减少了数据存储.

另外, 蒸馏算法在进行算符构造时, 只需要对本征矢量的内积进行操作, 不需要去更改约化传播子. 这意味着, 只需要计算一次约化传播子, 便可以

重复地利用; 配合上对本征矢量的各种操作, 就可以构造各种各样的算符. 这种灵活性和复用性是传统方法不具备的; 而且非连通部分的计算可以直接利用连通部分的中间结果, 不需要进行额外的计算. 因此, 蒸馏算法具有很多传统方法无法企及的优势.

虽然蒸馏算法有很多优点, 但是组合约化传播子和拉普拉斯算符本征系统来构建关联函数过程的计算复杂度正比于 $N_{\rm v}^4\times N_{\rm t}^2\times 4^4$. 将 $N_{\rm v}=10$ 和 $N_{\rm v}=100$ 进行对比, $N_{\rm v}=100$ 的计算复杂度是 $N_v=10$ 的 10000 倍. 而越是精确的计算越是要求 $N_{\rm v}$ 要足够大, 因此并行计算以及相应的优化是很有必要的. 我们根据蒸馏算法的特性, 开发实现了一套利用约化传播子和拉普拉斯算符本征系统计算关联函数的程序, 并进行了多方面的计算优化和测试.

本文将探讨蒸馏算法中利用约化传播子和拉普拉斯算符本征系统计算夸克双线性算符的关联函数的实现方法, 结合算法的计算特点提出了代码优化的设计方法, 以提高关联函数的计算效率. 首先介绍通过蒸馏算法构造关联函数的基本原理; 其次介绍具体的程序优化方案; 在第 3 部分, 给出了测试结果和分析; 最后在第 4 部分进行总结.

2 核心算法

为了降低问题的复杂性, 以关联函数的连通部分为例, 解释如何利用蒸馏算法计算关联函数并且优化程序. 非连通部分的关联函数的计算可以利用连通部分的计算中间结果得到, 因此只对连通部分进行讨论.

在格点 QCD 计算中, 通过蒸馏算法构造介子连通部分关联函数的形式[16] 为

$$\boldsymbol{C}^{A,B}(t',t)=\sum_{c_1,c_2,c_3,c_4=1}^{N_{\rm v}}\sum_{\alpha,\gamma,\kappa,\beta=1}^{4}\left[\varPhi^{A}{}_{\alpha\beta}^{c_1c_2}(t')\mathrm{e}^{-\sigma\lambda_{c_2}}\tau_{\beta\gamma}^{c_2c_3}(t',t)\mathrm{e}^{-\sigma\lambda_{c_3}}\varPhi^{B}{}_{\gamma\kappa}^{c_3c_4}(t)\mathrm{e}^{-\sigma\lambda_{c_4}}\tau_{\kappa\alpha}^{c_4c_1}(t,t')\mathrm{e}^{-\sigma\lambda_{c_1}}\right],\quad(6)$$

其中 c_1,c_2,c_3,c_4 均为对拉普拉斯算符本征值和本征矢量序号的标记. $N_{\rm v}$ 表示本征矢量的个数. $\alpha,\beta,\gamma,\kappa$ 均为物理学中的旋量指标, 它们的取值范围为 $\{0,1,2,3\}$. A,B 标记着物理上的不同算符, 每两个具有相同量子数的算符就可以用来构造关联函数, 它们会在不同的哈密顿量本征态上有不同的投影, 对应着不同的能量. 记关联函数矩阵[21] 为

$$\begin{bmatrix} C^{11} & C^{12} & \cdots & C^{1N_{\rm op}} \\ C^{21} & C^{22} & \cdots & C^{2N_{\rm op}} \\ \vdots & \vdots & \ddots & \vdots \\ C^{N_{op}1} & C^{N_{\rm op}2} & \cdots & C^{N_{\rm op}N_{\rm op}} \end{bmatrix},$$

$N_{\rm op}$ 表示关联函数矩阵的维数. (6) 式中 $\boldsymbol{\varPhi}$ 和 $\boldsymbol{\tau}$ 的表达式为

$$\boldsymbol{\tau}_{\alpha\beta}^{c_1c_2}(t,t')=\sum_{\boldsymbol{x},\boldsymbol{\omega}}\sum_{a,b}\boldsymbol{V}_{c_1}^{\dagger a}(\boldsymbol{\omega},t)\boldsymbol{G}_{\alpha\beta}^{ab}(t,\boldsymbol{w};t',\boldsymbol{x})\times\boldsymbol{V}_{c_2}^{b}(\boldsymbol{x},t'),\quad(7)$$

$$\boldsymbol{\varPhi}^{A}{}_{\alpha\beta}^{c_1c_2}(t)=\sum_{\boldsymbol{y},\boldsymbol{z}}\sum_{a,b}\boldsymbol{V}_{c_1}{}^{\dagger a}(\boldsymbol{y},t)\varGamma_{\alpha\beta}^{A,ab}(t,\boldsymbol{y},\boldsymbol{z})\boldsymbol{V}_{c_2}^{b}(t,\boldsymbol{z}),\quad(8)$$

$\boldsymbol{x},\boldsymbol{\omega}$ 均为空间指标. 通常 $\boldsymbol{\tau}$ 被称为约化传播子, 它是拉普拉斯算符本征空间的传播子, 对应传统算法的传播子. a,b,c,d 标记的是物理学中的色指标, 它的取值范围是 $\{0,1,2\}$. $\boldsymbol{\varPhi}$ 的不同定义决定了算符的量子数, 对应到真实世界中的不同粒子. 实际上, (7) 式中的 $\boldsymbol{G}$ 就是传统方法中的传播子, 可以看到由于本征矢量与传播子的空间指标和色指标进行了求和收缩合并, 使得 $\boldsymbol{\tau}$ 和 $\boldsymbol{\varPhi}$ 只剩下了表示本征矢量的指标、时间的指标和旋量的指标. 注意 $\mathrm{e}^{-\sigma\lambda_{c_1}}$ 这一项由格点 QCD 中的涂摩算法给出. λ_{c_1} 表示第 c_1 个本征矢量的本征值, 其值越大, 该本征矢量对关联函数的贡献就越小, 因此通常选取 $N_{\rm v}$ 个最小的本征值对应子空间来近似计算全传播子.

通过蒸馏算法计算关联函数的步骤如图 1, 首先解拉普拉斯算符的本征值和本征矢量; 再利用本征矢量, 计算 $\boldsymbol{\tau}$, 并保存以便重复使用; 根据研究的需要利用本征矢量和本征值构造 $\boldsymbol{\varPhi}$; 最后按照 (6) 式, 利用 $\boldsymbol{\varPhi}$ 和 $\boldsymbol{\tau}$ 计算相应的关联函数. 本文工作主要涉及后两步. 计算 λ, $\boldsymbol{\nu}$, $\boldsymbol{\tau}$ 的过程只需进行一次. 在后续的计算中根据不同的研究需要, 对本征矢 $\boldsymbol{\nu}$ 进行相应的操作再与约化传播子 τ 进行缩并可以构造出相应的关联函数. 重复利用 λ, $\boldsymbol{\nu}$, $\boldsymbol{\tau}$. 相较于传统方法在每次进行算符构造时都需要进行传播子求解, 蒸馏算法构造算符更加方便, 在有了约化传播子和本征值、本征矢量后, 计算关联函数更经济. 同时, 由于产生约化传播子的过程与构造算

符的过程相对独立, 蒸馏算法可以在研究课题没有确定的情况下, 不断利用可用的计算资源产生约化传播子和本征值、本征矢量, 合理利用资源.

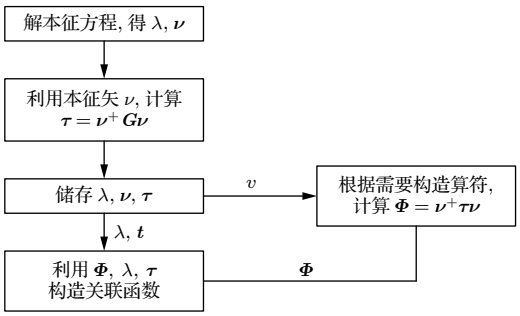

图 1 利用蒸馏算法计算关联函数的流程

Fig. 1. The procedure of computing correlators via distillation method.

(8) 式中 $\boldsymbol{\Phi}$ 的计算实际上可以分解为两部分, $\boldsymbol{\Phi}=\boldsymbol{\Phi}_{\rm C}\otimes\boldsymbol{\Phi}_{\rm G}$. $\boldsymbol{\Phi}_{\rm C}$ 代表拉普拉斯本征空间的计算, $\boldsymbol{\Phi}_{\rm G}$ 代表旋量空间相关的计算. $\boldsymbol{\Phi}_{\rm C}$ 和 $\boldsymbol{\Phi}_{\rm G}$ 的计算是互相独立的, 并且 $\boldsymbol{\Phi}_{\rm G}$ 是一些常数矩阵的组合, 不需要额外进行计算. 在程序计算时, 将 $\boldsymbol{\Phi}_{\rm G}$ 固定在程序内部, 可变的 $\boldsymbol{\Phi}_{\rm C}$ 由外部文件读入, 然后在根据需要将 $\boldsymbol{\Phi}_{\rm C}$ 和 $\boldsymbol{\Phi}_{\rm G}$ 通过一定方式组合起来得到所需的具有确定量子数的 $\boldsymbol{\Phi}$.

本征矢量的个数 $N_{\rm v}$ 的取值影响着物理信号, 通常需要 $N_{\rm v}$ 足够大以便于对尽可能多的情况保持良好的信号, 而蒸馏算法从约化传播子计算关联函数的复杂度 $\propto N_{\rm v}^4$. 虽然相比传统方法计算一个全传播子的关联函数的计算量要小很多, 但仍然需要处理 $N_{\rm op}^2\times N_{\rm v}^4\times 4^4\times N_{\rm t}^2$ 次乘法. 通常 $N_{\rm op}$ 的取值是 $\mathcal{O}(10)$ 级别, N_v 是 $\mathcal{O}(100)$ 级别, $N_{\rm t}$ 也是 $\mathcal{O}(100)$ 级别.

为加速计算, 在进程级别、线程级别和指令集级别分别进行了算法的计算优化.

3 优化方案

对 (6) 式进行计算约化, 以减少不必要的计算开销. 结合蒸馏算法计算的特性, 选择时间维度上的切片操作, 切割后的数据处理相对独立, 使用 MPI 多线程方式实现时, 避免了数据通信成为瓶颈的可能; 另外高维约化传播子和 $\boldsymbol{\Phi}$ 的数据被划分到多个计算节点上处理, 也避免了单个计算节点的内存容量限制. 为了进一步提高计算并行度, 在 MPI 进程中, 实现了基于共享内存特性的线程级并行计算, 最大可能地提升了计算节点中资源的利用率. 最后, 由于涉及到大量复杂的复数运算操作, 开展了数据结构的矢量化适配, 实现了 SIMD 指令级的并行计算.

3.1 计算约化

前面已经提到, 在蒸馏算法中计算量 $\propto N_{\rm op}^2\times N_{\rm v}^4$. 我们发现通过一些公式变形, 可以将这个计算量约化成 $\propto N_{\rm op}\times N_{\rm v}^3$.

关联函数计算表达式 (6) 式变形成

$$\begin{aligned}\boldsymbol{C}^{A,B}(t,t')=&\sum_{c_1,c_3=1}^{N_{\rm v}}\sum_{\alpha,\gamma=1}^{4}\left(\sum_{c_2=1}^{N_{\rm v}}\sum_{\beta=1}^{4}[\boldsymbol{\Phi}^{A}_{\alpha\beta}{}^{c_1c_2}(t')\boldsymbol{\tau}^{c_2c_3}_{\beta\gamma}(t',t)]\right)\\&\times\left(\sum_{c_2=1}^{N_{\rm v}}\sum_{\beta=1}^{4}\boldsymbol{\Phi}^{B}_{\gamma\beta}{}^{c_3c_2}(t)\boldsymbol{\tau}^{c_2c_1}_{\beta\alpha}(t,t')]\right),\end{aligned}\tag{9}$$

注意到括号中的 c_2 和 β 指标下的部分计算用 $\boldsymbol{T}$ 矩阵表示, 那么关联函数可以写成

$$\boldsymbol{C}^{A,B}(t',t)=\sum_{c_1,c_3=1}^{N_{\rm v}}\sum_{\alpha,\gamma=1}^{4}\boldsymbol{T}^{A}_{\alpha\gamma}{}^{c_1c_3}\boldsymbol{T}^{B}_{\gamma\alpha}{}^{c_3c_1}.\tag{10}$$

在计算关联函数之前, 可以先计算出 $\boldsymbol{T}$ 矩阵, 然后利用它来计算不同的关联函数. 因为 $\boldsymbol{T}$ 的计算量 $\propto N_{\rm op}\times N_{\rm v}$, 于是关联函数的计算量 $\propto N_{\rm op}\times N_{\rm v}^3$, 可见计算复杂度比原来小很多, 在 $N_{\rm v}$ 和 $N_{\rm op}$ 越大时越明显. 实现计算约化后的程序流程如图 2.

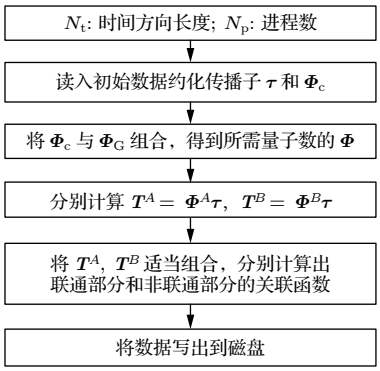

图 2 含计算约化的关联函数计算的流程. 其中 $\boldsymbol{T}^A$ 和 $\boldsymbol{T}^B$ 表示两个中间计算量. 利用中间量的计算减少了总体的计算量, 让计算量从 $\propto N_{\rm op}^2\times N_{\rm v}^4$ 变成 $\propto N_{\rm op}\times N_{\rm v}^3$, 极大地减少了计算量

Fig. 2. The flowchart of computing correlation function. $\boldsymbol{T}^A$ and $\boldsymbol{T}^B$ are two intermidiate quantities. After introducted intermediate quantities, the computation consumption is highly reducted to $\propto N_{\rm op}\times N_{\rm v}^3$.

3.2 进程和线程级优化

注意到关联函数的计算关于各个时间片 t 和 t' 独立, 因此选择对时间维度进行切分, 这样就能够在各个进程进行独立的计算, 完成各个 t, t' 所负责的那部分数据进行收缩合并, 无需考虑通信开销. 程序实现是选择 MPI 并行计算方式, 每个 MPI 进程负责各个独立的时间片, 在 MPI 进程中进一步启用 OpenMP 线程共享内存的并行计算, OpenMP 多线程没有跨计算节点的通信, 因此不会增加节点间通信.

在程序实现时, 选择依次对 t 和 t' 切分. 先对 t 进行切分, 当进程数大于 $N_{\rm t}$ 时再考虑对 t' 进行切分. 以 $N_t = 128$ 为例, 如果进程数等于 64, 那么第一个进程就要计算 $t = 0, 1$, $t' = 0, 1, \cdots, 127$ 的关联函数, 第二个进程就要计算 $t = 2, 3$, $t' = 0, 1, \cdots, 127$ 的关联函数的数据. 如果进程数等于 256, 那么第一个进程就要计算 $t = 0$, $t' = 0, 1, \cdots, 63$ 的关联函数, 第二个进程就要计算 $t = 0$, $t' = 64, 65, \cdots, 127$ 的数据, 以此类推. 这样进行划分的另外一个优势是, 由于关联函数具有随 t' 指数衰减的性质, 以及对于 t 具有平移不变性, 可以利用这两个性质来检查计算结果的正确性.

由于约化传播子 $\boldsymbol{\tau}$ 以及 $\boldsymbol{\Phi}$ 的维度 $\propto N_{\rm v}^2$, 当 $N_{\rm v}$ 的取值接近 1000, $N_{\rm t} = 128$ 时, 按双精度存储, $\boldsymbol{\tau}$ 的数据量大约 1 TB, 因此在 MPI 实现时使用并行 IO 的数据读入方式, 并行 IO 也是按时间片划分方式, 和 MPI 并行计算方案保持一致. 注意到 $\boldsymbol{\Phi}$ 只有一个时间指标, 而 $\boldsymbol{\tau}$ 有两个时间指标. 在实际计算中, 当并行进程数 $N_{\rm p} \leqslant N_{\rm t}$ 时, 只对 $\boldsymbol{\tau}$ 其中一个时间指标进行切分, 分配到各个进程上去, 而 $\boldsymbol{\Phi}$ 不切分; 当并行进程数 $N_{\rm p} > N_{\rm t}$ 时, 同时对 $\boldsymbol{\tau}$ 的两个时间维度进行切分, 而切片 $\boldsymbol{\tau}$ 的计算只依赖部分 $\boldsymbol{\Phi}$, 即每个并行进程上都只需 $\boldsymbol{\Phi}$ 的部分矩阵元, 因此 $\boldsymbol{\Phi}$ 切片依赖于 $\boldsymbol{\tau}$ 的切分方式. 其流程如图 3 所示.

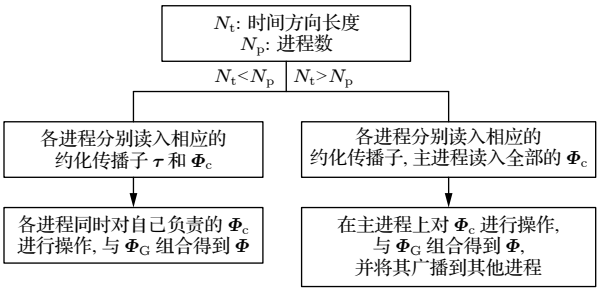

图 3 按照时间切分实现并行计算的方式, 根据 $N_{\rm p}$ 与 $N_{\rm t}$ 的相对大小, 由于数据的特性, 对 $\boldsymbol{\tau}$ 和 $\boldsymbol{\Phi}$ 按情况采用不同的切分方法.

Fig. 3. Data sgemented according to time. Two conditions are considered which decided how $\boldsymbol{\tau}$ and $\boldsymbol{\Phi}$ are treated because of the feature of data.

3.3 指令级优化

在计算中, 考虑到形如

$$\boldsymbol{T}^{Ac_1c_3} = \sum_{c_2=1}^{N_{\rm v}} \sum_{\beta=1}^{4} \left[\boldsymbol{\Phi}_{\alpha\beta}^{Ac_1c_2}(t') \boldsymbol{\tau}_{\beta\gamma}^{c_2c_3}(t', t) \right], \tag{11}$$

$$\boldsymbol{C}^{A,B}(t, t') = \sum_{c_1,c_3=1}^{N_{\rm v}} \sum_{\alpha,\gamma=1}^{4} \boldsymbol{T}_{\alpha\gamma}^{Ac_1c_3}(t', t) \boldsymbol{T}_{\gamma\alpha}^{Bc_3c_1}(t, t') \tag{12}$$

的张量缩并过程, 可以将其简化为与缩并指标相对应的矩阵乘法. 上述的两个表达式都可以看作形如 $(4N_{\rm v}, 4N_{\rm v})$ 的矩阵乘法. 如将原本按照 $(4, 4, N_{\rm v}, N_{\rm v})$ 行优先存储的数据转换为 $(4N_{\rm v}, 4N_{\rm v})$ 行优先存储的格式, 那么可得如下的计算方式:

$$(\boldsymbol{AB})_{ij} = \sum_{k=1}^{4N_{\rm v}} \boldsymbol{A}_{ik} \boldsymbol{B}_{kj}. \tag{13}$$

行优先存储格式下按 (13) 式作矩阵乘法时, $\boldsymbol{B}$ 矩阵无法实现连续的内存访问, 因此将 $\boldsymbol{B}$ 矩阵转置, 计算变为

$$(\boldsymbol{AB})_{ij} = \sum_{k=1}^{4N_{\rm v}} \boldsymbol{A}_{ik} \boldsymbol{B}_{jk}^{\rm T}, \tag{14}$$

则计算中在循环 k 时可以内存连续地访问矩阵的元素.

在计算中, 每个数字实际上是复数, 而由于复数乘法规则复杂, 在不更改数据排列的情况下, 编译器无法保证利用 SIMD 矢量运算提升复数计算性能. 因此将矩阵的实部和虚部分离存储, 利用运算法则将 1 个复矩阵的乘法分解为 4 个实矩阵乘法和 2 个实矩阵加法, 形式为 $\mathrm{Re}(\boldsymbol{AB}) = \mathrm{Re}(\boldsymbol{A})\mathrm{Re}(\boldsymbol{B}) - \mathrm{Im}(\boldsymbol{A})\mathrm{Im}(\boldsymbol{B})$, $\mathrm{Im}(\boldsymbol{AB}) = \mathrm{Re}(\boldsymbol{A}) \times \mathrm{Im}(\boldsymbol{B}) + \mathrm{Im}(\boldsymbol{A})\mathrm{Re}(\boldsymbol{B})$.

在上述调整下, 变量的内存访问具有连续性, 有助于编译器自动完成 SIMD 矢量计算优化. 通过确认编译得到的汇编代码, 编译器确实启用 SIMD 矢量运算指令. 当然, 也根据测试平台手动调用 SIMD 相关函数和指令实现了计算, 经过测试相比编译器自动优化提升不到 10%, 考虑到硬编码的繁

琐之处和兼容问题, 我们最终采用的方式是修改数据格式和实现复数计算后, 通过编译器自动优化完成 SIMD 向量化.

4 测试结果

为测试加速效果, 在 $N_{\rm t}=128$, $N_{\rm s}=16$ 的格子上进行测试. 组态的海里只包含两味粲夸克. 计算过程中采用双精度. $a_{\rm t}^{-1}=9.6$ Gev . 分别对 SIMD, MPI, OpenMP 的加速效果进行测试, 并对物理计算的结果进行展示.

4.1 SIMD 计算加速测试

为了验证 SIMD 计算加速效果, 选择一个含有 $N_{\rm v}=10$个本征向量的算例, 按照 (9) 式来计算关联函数矩阵 (5×5维). 测试的处理器支持 FMA 和 AVX SIMD 指令集. 该 CPU 处理器包含 8 个物理核心, 可通过超线程技术实现的 16 进程并行. 计算时对整个流程按步骤分别计时, 具体步骤包括: 初始化、数据 IO 读取、计算 $\boldsymbol{T}$ 中间量、利用 $\boldsymbol{T}$ 矩阵计算关联函数. 为了实现使用 SIMD 优化效果, 还对照测试了直接调用标准库中的复数计算函数的程序设计. 按步骤计时的结果如图 4, 使用 SIMD 前后计算性能如图 5.

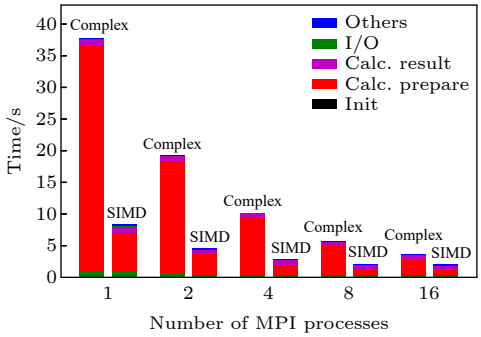

图 4 使用 SIMD 优化前后各阶段耗时对比. I/O 代表图 1 中第一步和第二步的时间, Calc.prepare 代表图 1 中第三步的时间, Calc.result 代表图 1 中第四步的时间, Others 代表图 1 中第五步的时间, Init 代表程序初始化的时间. 图例 SIMD 表示启用了 AVX 形式的 SIMD 计算性能, 而 Complex 表示程序直接调用标准库中的复数计算函数 (此处未使用 SIMD 计算). 其中 16 个 MPI 进程并行计算的结果是在超线程计算状态下获得

Fig. 4. The cost of time of program's each part to see the effects of SIMD. I/O labels the time of first step and second step in Fig. 1, Calc.prepare labels the time of the third step in Fig. 1, Calc.result labels the time of the fourth step in Fig. 1, Others labels the time of the fifth step in Fig. 1. Init labels the time of initialization. SIMD in the picture means SIMD optimization was adopted and Complex in the picture means the stdandard library of complex computation was used. And hyper-threading technology was used for 16 MPI process.

程序在启用 SIMD 指令后, 在不考虑超线程的时, 运行时间变为原来的 1/4. Calc.prepare的时间变为原来的 1/6. Calc.prepare 是程序的热点, 此处计算程序中间量, 计算时间随 $N_{\rm v}$ 的变化最明显.

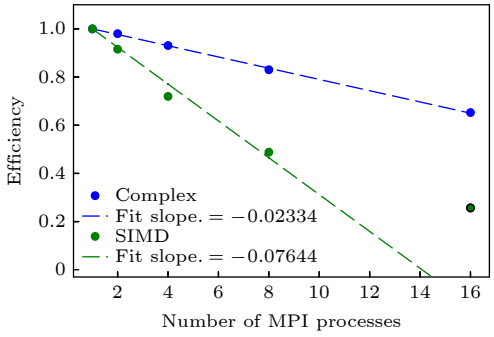

图 5 使用 SIMD 优化前后性能对比. 图例 SIMD 表示启用了 AVX 形式的 SIMD 计算性能, 而 Complex 表示程序直接调用标准库中的复数计算函数 (此处未使用 SIMD 计算). 其中, 在 SIMD 启用时 16 个超线程计算结果未参与数据拟合

Fig. 5. The cost of time of program's each part to see the effects of SIMD. SIMD in the picture means SIMD optimization was adopted and Complex in the picture means the stdandard library of complex computation was used. And hyper-threading technology was used for 16 MPI process.

4.2 OpenMP 计算加速测试

现测试 OpenMP 计算加速效果, 沿用了上一节测试的算例, 处理器开启 CPU 的超线程, 并关闭 SIMD 加速. 同样按步骤计时的测试结果如图 6.

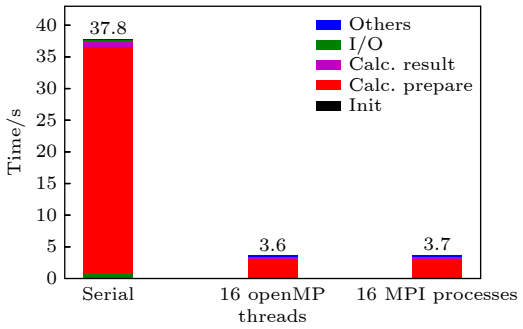

图 6 使用 OpenMP 优化前后耗时对比. 图例如图 4. 图例 Serial 表示串行版本, 即未开启 OpenMP 多线程和 MPI 多进程

Fig. 6. The effects of OpenMP optimization was showed. Legends are the same as 4. Serial lables the results of serial program which means no OpenMP and MPI was adopted.

从结果可见, 程序采用 OpenMP 线程加速方式有性能提升, 使用 16 个超线程加速相对单核获得了约 10 倍的加速. 同时采用 OpenMP 线程的计算效率与采用 MPI 进程的计算效率几乎相同.

4.3 MPI 强扩展性测试

为了测试 MPI 并行计算的强扩展性, 选择一个含有 $N_{\rm v}=100$ 个本征向量的大算例, 此时输入文件体积达到 40 GB, 采用不同的计算核数分别按照 (9) 式来计算关联函数矩阵 (5×5维). 测试的处理器支持 FMA 和 AVX SIMD 指令集, 最大 MPI 进程达到 512 个. 同样按初始化、数据 IO 读取、计算 $\boldsymbol{T}$ 中间量、利用 $\boldsymbol{T}$ 矩阵计算关联函数等步骤计时. 结果如图 7. MPI 并行的效果明显, 适合大规模并行. MPI 进程数的增加, 效果体现在 Calc.prepare 部分, 也就是 (11) 式中间量的计算加速效果最明显. 由于各部分加速的效果不同, 当 512 个进程时, Calc.prepare 的计算时间与 I/O 的时间差别不大. I/O 随进程数的变化时间改变并不明显, 耗时可以忽略.

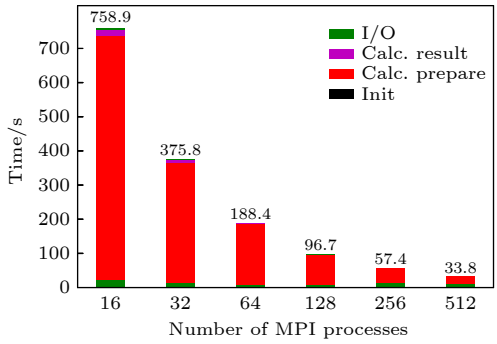

图 7 MPI 并行强扩展性测试. 随着 MPI 进程数增加, 计算时间成比例减少. 图例如图 4

Fig. 7. MPI parallelism in strong scale tests. The cost time decrease with MPI process numbers. Legends are the same as Fig. 4.

计算程序在强扩展性测试中的并行计算效率如图 8, 从结果可见, 总体并行计算效率很高, 在 512 个进程的情况下仍能达到 70% 左右的效率. 因此程序适合大规模计算加速.

4.4 MPI 弱扩展性测试

为了测试 MPI 并行计算的弱扩展性, 选择几个不同规模的算例, 其进程数分别为 $N_{\rm p}=2,16,128,1024$, 关联函数矩阵维度仍然为 5×5. 测试的

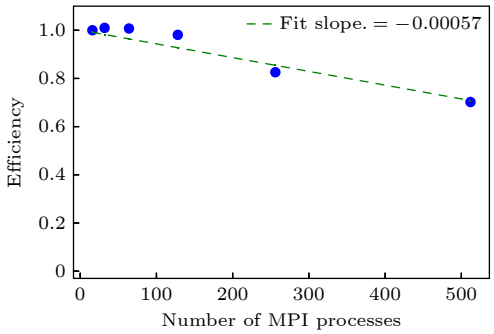

图 8 MPI 并行强扩展性测试, 不同 MPI 进程数的测试相对于 16 个进程的计算效率

Fig. 8. MPI parallelism in strong scale tests. The efficiency of strong scale tests compared with 16 MPI processes.

处理器支持 FMA 和 AVX SIMD 指令集, 对不同算例等比例使用不同规模的并行进程数, 使得理论上每个进程的负载相同. 按步骤计时的结果如图 9, 忽略 I/O 部分, 其他部分表现良好. 但 I/O 部分当进程数等于 1024 时, 耗时出现了明显的增加, 并与热点部分 (Calc.prepare) 的耗时可比. 这受限于硬件的性能. 在当前测试中, 当 $N_{\rm p}\leqslant128$ 时程序在弱扩展性测试中的并行效率较高, 但 $N_{\rm p}=1024$ 的大规模测试中展现出了一个因系统资源限制产生的问题, 由于磁盘带宽是一定的, I/O 请求数激增, I/O 部分时间大大增加, 甚至超过计算中间变量 $\boldsymbol{T}$ 矩阵 (Calc.prepare) 部分成为了新的热点; 但并行 I/O 能够解决串行读入数据时内存不足的问题. 由于计算量是随着 $N_{\rm v}$ 的三次方增加的, 而数据量大小是随着 $N_{\rm v}$ 平方增加的, 这会导致如果要维持每个进程的计算量不变相应的数据量就会变少, 读入的数据会随着 $N_{\rm v}$ 的增加而变得零碎, 最终受限于 I/O.

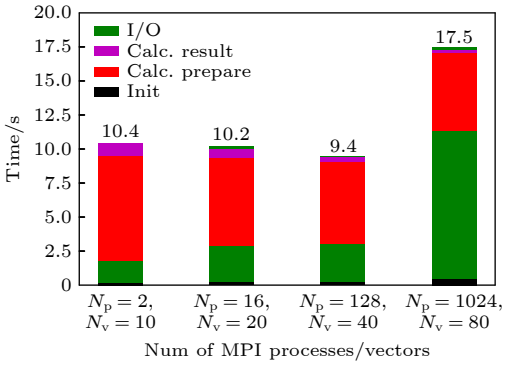

图 9 MPI 并行弱扩展性测试. $N_{\rm p}$ 表示使用的并行进程数, $N_{\rm v}$ 表示本征向量数. 图例说明同图 4

Fig. 9. MPI parallelism in weak scale tests. $N_{\rm p}$ represents the number of process, $N_{\rm v}$ represents the number of eigenvectors. Legends are the same as Fig. 4.

4.5 物理结果

为了检验结果的正确性, 对物理结果进行了测试. 构造了三个 $J^{\rm PC}$ 量子数为 0^{-+} 的算符, 分别计算它们的关联函数并分析有效质量, $m_{\rm eff}(t) = -\ln\dfrac{C(t+1)}{C(t)}$. 这里 $C(t)$ 表示某种量子数算符的关联函数. 通过比较这三个算符所得的关联函数的有效质量, 在时间片比较大的时候趋于同一个平台, 与传统方法所得结果一致, 见图 10.

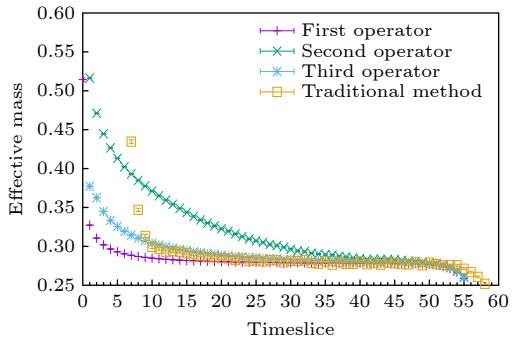

图 10 做变分前的结果. 在时间比较小时, 三个算符得到的有效质量的行为有很大差别, 证明在不同态的投影是不同的, 意味着变分会有一定的效果. 在时间较大时, 三个算符的有效质量趋于同一平台, 说明它们的量子数是相同的, 可以用来变分. traditional method 表示第一个算符通过传统方法所得到的有效质量, 作为蒸馏算法的参照

Fig. 10. Results before variation. The behaviors of the effective mass of these three operators are very different and it means variational analysis would give good results. When time is large enough, these three operators approach to the same plateau so that they should have the same quantum numbers. traditial method label the effective mass of first operator throgh traditional mehtod, which can is matched with distillation method.

因为三个算符在不同本征能量态上的投影有很大不同, 如果中间计算没有错误, 通过变分分析 [21] 可以近似得到三个主要投影到最低的三个能量本征态的算符, 变分方程如下:

$$\sum_B \boldsymbol{C}^{A,B}(t)\boldsymbol{V}_{iB} = \sum_B {\rm e}^{-m_i(t-t_0)}\boldsymbol{C}^{A,B}(t_0)\boldsymbol{V}_{iB}, \tag{15}$$

变分后, 可以得到本征矢 $\boldsymbol{V}_i$, 对应第 i 个态 m_i, 将 $\boldsymbol{C}^{A,B}$ 与 $\boldsymbol{V}_i$ 组合, 可以得到对应到相应态的关联函数

$$\boldsymbol{C}_i(t) = \sum_{A,B} \boldsymbol{V}_{iA}\boldsymbol{C}^{A,B}(t)\boldsymbol{V}_{iB}, \tag{16}$$

并且利用这三个算符所计算出的有效质量要能够相互区别开, 在时间片比较大的时候趋于不同的平台. 因此可以此来验证计算的正确性.

利用上述三个算符构造关联函数矩阵, 做变分分析, 得到如图 11 所示的结果. 变分后得到三个不同的态, 分别对应着基态、第一激发态、第二激发态. 利用变分本征矢量组合后的关联函数所得的有效质量在时间较大时, 分别趋于不同的平台, 证明变分将原来的三个算符分别组合成了三个主要投影到最低三个态的优化算符. 第二激发态的信号略差. 变分效果明显, 再次验证了结果的正确性.

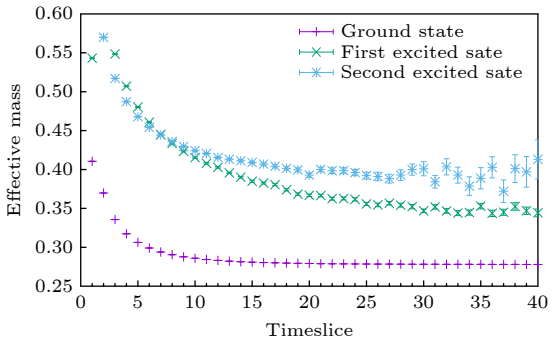

图 11 做变分后的结果

Fig. 11. Results after variation.

5 结论与展望

本文围绕格点 QCD 蒸馏算法中关联函数的计算优化问题, 提出了一套加速程序计算的有效解决方案, 结合理论分析采用了 MPI, OpenMP 和 SIMD 多级别的计算优化技术. 文中首先介绍了蒸馏算法中关联函数计算中各个模块的优化设计, 并对程序的性能进行了多方面的分析和测试. 实验结果表明蒸馏算法计算关联函数效率的显著提升, 并且验证了物理结果的正确性.

参考文献

[1] Flynn J M, Mescia F, Tariq A S B 2003 *JHEP* **07** 066
[2] Lozano J, Agadjanov A, Gegelia J, Meißner U G, Rusetsky A 2021 *Phys. Rev. D* **103** 034507
[3] Chen C, Fischer C S, Roberts C D, Segovia J 2021 *Phys. Lett. B* **815** 136150
[4] Meißner U G 2014 *Nucl. Phys. News.* **24** 11
[5] Lähde T A, Meißner U G 2019 *Lect. Notes Phys.* **957** 1
[6] Wilson K G 1974 *Phys. Rev. D* **10** 2445
[7] Gasser J, Leutwyler H 1984 *Annals Phys.* **158** 142
[8] Diakonov D, Petrov V, Pobylitsa P, Polyakov M V, Weiss C 1996 *Nucl. Phys. B* **480** 341
[9] Rothe H J 2012 *World Sci. Lect. Notes Phys.* **82**

[10] Brower R, Christ N, DeTar C, Edwards R, Mackenzie P 2018 *EPJ Web Conf.* **175** 09010

[11] Zhang Z, Luan Z, Xu C, Gong M, Xu S 2018 *2018 IEEE Intl Conf on Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications* (*ISPA/IUCC/BDCloud/SocialCom/SustainCom*),Melbourne, VIC, Australia 605

[12] Gattringer C, Lang C B 2010 *Lect. Notes Phys.* **788** 1

[13] Barrett R, Berry M, Chan T F, Demmel J, Donato J M, Dongarra J, Eijkhout V, Pozo R, Romine C, Vorst H V 1994 *SIAM, Philadelphia* 139, 140, 141

[14] Press W H, Teukolsky S A, Vetterling W T, Flannery B P 1999 (Cambridge: Cambridge University Press) p139

[15] Wilcox W, Darnell D, Morgan R, Lewis R 2006 *PoS LAT* **2005** 039

[16] Peardon M, Bulava J, Foley J, Morningstar C, Dudek J, Edwards R G, Joó B, Lin H W, Richards D G, Juge K J 2009 *Phys. Rev. D* **80** 054506

[17] Egerer C, Edwards R G, Orginos K, Richards D G 2021 *Phys. Rev. D* **103** 034502

[18] Güsken S, Löw U, Mütter K H, Sommer R 1989 *Phys. Lett. B* **227** 266

[19] Best C, et al. 1997 *Phys. Rev. D* **56** 2743

[20] Basak S, Edwards R G, Fleming G T, Heller U M, Morningstar C, Richards D, Sato I, Wallace S 2005 *Phys. Rev. D* **72** 094506

[21] Ehmann C, Bali G 2007 *PoS LATTICE* **2007** 094

Calculation and optimization of correlation function in distillation method of lattice quantum chromodynamcis*

Zhang Ren-Qiang [1)5)] Jiang Xiang-Yu [1)5)] Yu Jiong-Chi [2)] Zeng Chong [3)]
Gong Ming [1)5)] Xu Shun [4)†]

1) (*Theoretical Physics Division, Institute of High Energy Physics, Chinese Academy of Sciences, Beijing* 100049, *China*)

2) (*Department of Polymer Science and Engineering, Zhejiang University, Hangzhou* 310058, *China*)

3) (*College of Computer Science and Technology, Zhejiang University, Hangzhou* 310058, *China*)

4) (*Computer Network Information Center, Chinese Academy of Sciences, Beijing* 100190, *China*)

5) (*School of Physical Science, University of Chinese Academy of Sciences, Beijing* 100049, *China*)

(Received 6 January 2021; revised manuscript received 26 March 2021)

Abstract

Lattice quantum chromodynamics (lattice QCD) is a theory based on quantum chromodynamics, which is widely used in strong interaction related calculations. As a research method that can give accurate and reliable theoretical results, with the improvement of computer ability, Lattice QCD is playing an increasingly important role in recent years. Distillation method is an important numerical method to calculate hadron correlation function in lattice QCD, and can improve the signal-to-noise ratio of calculated physical quantities. Distillation is a method to approximately compute full propagator via replace the laplacian operator with it's outerproduct of laplace eigenvectors. In this way, the construction of operators is independent of the inversion of propagator which is costful. The eigenvector system and perambulator can be used in different physical projects and we don't need to compute these data repeatedly. It's also convinent for computing disconnected part of correlation function. However, it also faces to the problem of large amount of data in constructing correlation function because the difficulty of compuation is proportional to the cubic of the number of eigenvectors, so it is necessary to further improve its computational efficiency. A program is developed in this work to construct correlation function of quark bilinear with distillation method, and solved the bottleneck of computing performance by using MPI(Message Passing Interface, https://www.open-mpi.org), OpenMP(Open Multi-Processing) and SIMD(Single Instruction Multiple Data) multi-level optimization technology. And this program distribute timeslices to different MPI processes because the computation of each timeslice is independent. In order to show the efficiency of our program some tests result are presented. After various tests of the program, it shows that our design can support large-scale computation. Under the strong scalability test, the parallel computing efficiency of 512 processes can still achieve about 70%. The ability of calculating correlation function is greatly improved. The correction of results also has been checked via compute pseudo-scalar correlators of charmonium. Three different 0^{-+} operators were adopted for variational analysis and there effecitive mass plateau were compared with the effective mass obtained from the tradional method with point source. The results of distillation method are consistent with traditional method. After variational analysis, three state is obtained, which means the variational analysis take effects and the correlation functions obtained from distillation method is reasonable.

Keywords: lattice quantum chromodynamics, distillation algorithm, correlation function, parallel computing

PACS: 12.38.Gc **DOI:** 10.7498/aps.70.20210030

* Project supported by the National Key R&D Program of China (Grant No. 2017YFB0203203) and the National Natural Science Foundation of China (Grant Nos. 11775229, 11935017).

† Corresponding author. E-mail: xushun@sccas.cn