计算机科学
COMPUTER SCIENCE

基于国产c86处理器的CP2K软件移植与优化

范黎林, 乔一航, 李俊飞, 柴旭清, 崔容培, 韩秉豫

引用本文

范黎林, 乔一航, 李俊飞, 柴旭清, 崔容培, 韩秉豫基于国产c86处理器的CP2K软件移植与优化[J]. 计算机科学, 2023, 50(6): 58-65.

FAN Lilin, QIAO Yihang, LI Junfei, CHAI Xuqing, CUI Rongpei, HAN Bingyu. CP2K Software Porting and Optimization Based on Domestic c86 Processor [J]. Computer Science, 2023, 50(6): 58-65.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.230200213

基于国产 c86 处理器的 CP2K 软件移植与优化

范黎林[1,2,3]　乔一航[1,2,3]　李俊飞[4]　柴旭清[1,2,3]　崔容培[1,2,3]　韩秉豫[2,3,5]
1 河南师范大学计算机与信息工程学院　河南 新乡 453007
2 河南师范大学高性能计算中心　河南 新乡 453007
3 智慧商务与物联网技术河南省工程实验室　河南 新乡 453007
4 中国科学院大学计算机科学与技术学院　北京 100049
5 河南师范大学软件学院　河南 新乡 453007
(fanlilin@htu.edu.cn)

摘　要　CP2K 是目前运行最快的开源第一性原理材料计算和模拟软件,源码中调用协处理器的部分基于 CUDA 架构编写。因平台底层硬件架构和编译环境不同,原生的 CP2K 软件无法调用国产 c86 处理器平台上的 DCU,因此不能实现跨平台应用。为解决该问题,提出了一种 CP2K 面向该平台的移植方案。该方案的核心思想为:对 CP2K 软件中主要基于 CUDA 接口实现的 DBCSR 库进行代码分析,拆解对应结构体和类的封装方式,并基于 HIP 的编程标准对其进行实现和封装。在国产 c86 处理器平台上编译安装 HIP 版的 DBCSR 库,链接 CP2K 软件,最终实现运行 DCU 版的 CP2K 软件。后续选取两个测试算例,基于编译级与运行级对其进行优化实验。实验发现,删除 CP2K 脚本链自动安装的 FFTW 库可提高计算结果精度。实验结果表明,所使用的优化方法可显著提升 CP2K 软件的计算效率和计算准确性,为实现开源软件面向国产平台的移植优化和国产化替代做出贡献。

关键词:CP2K;DBCSR;编译优化;MPI 运行优化;HIP 移植;JIT 编译
中图法分类号　TP391

CP2K Software Porting and Optimization Based on Domestic c86 Processor

FAN Lilin[1,2,3],QIAO Yihang[1,2,3],LI Junfei[4],CHAI Xuqing[1,2,3],CUI Rongpei[1,2,3] and HAN Bingyu[2,3,5]
1 School of Computer and Information Engineering,Henan Normal University,Xinxiang,Henan 453007,China
2 High Performance Computing Center,Henan Normal University,Xinxiang,Henan 453007,China
3 Engineering Laboratory of Intelligent Business and Internet of Things Technology,Xinxiang,Henan 453007,China
4 School of Computer Science and Technology,University of Chinese Academy of Sciences,Beijing 100049,China
5 College of Software,Henan Normal University,Xinxiang,Henan 453007,China

Abstract　CP2K is currently the fastest open source first-principles materials calculation and simulation software,and the part of the source code that calls the coprocessor is written based on the CUDA architecture. Due to the different underlying hardware architecture and compilation environment of the platform,the native CP2K software cannot call the DCU on the domestic c86 processor platform to achieve cross-platform applications. In order to solve this problem,a CP2K porting scheme for this platform is proposed. The core idea is to analyze the code of the DBCSR library mainly based on the CUDA interface in CP2K software,disassemble the encapsulation method of the corresponding structure and class,and implement and package it based on the programming standard of HIP. The DBCSR library of HIP version is compiled and installed on the domestic c86 processor platform,and the CP2K software is linked to finally realize the CP2K software running the DCU version. Then,two test studies are selected and optimized based on the compilation level and run-level. It is found that removing the FFTW library automatically installed by CP2K script chain can improve the accuracy of calculation results. Experimental results show that the optimized method used can

到稿日期:2023-02-27　返修日期:2023-04-11
基金项目:光合基金 B 类(2021070222021070227 68,2021070222021070226 86);河南省高等教育教学改革研究与实践立项项目 (2021SJGLX354);中国高校产学研创新基金-新一代信息技术创新项目资助课题计划书 (2020ITA07040);产学合作协同育人项目(202102089014,202102533043)
This work was supported by the Guanghe Fund Class B(2021070222021070227 68,2021070222021070226 86),Henan Province Higher Education Teaching Reform Research and Practice Project(2021SJGLX354),China University Industry-University-Research Innovation Fund-New Generation Information Technology Innovation Project Funding Project Plan(2020ITA07040) and Industry-University Cooperation and Education Project(202102089014,202102533043).
通信作者:柴旭清(cxq@htu.edu.cn)

significantly improve the computational efficiency and calculation accuracy of CP2K software, and contribute to the porting optimization and localization of open source software for domestic platforms.

Keywords CP2K, DBCSR, Compilation optimization, MPI running optimization, HIP transplantation, Just-in-time compilation

1 引言

分子动力学计算机模拟是一类重要的高性能计算应用，是生物、化学、材料学、力学等学科研究的重要工具[1]。在超算平台上使用模拟软件进行大规模分子动力学模拟已成为研究相关重大课题的必要手段[2-3]。CP2K 是一个量子化学和固态物理软件包，可对固态、液态、分子、周期性、材料、晶体和生物系统进行原子模拟，在学术界和工业界发挥着重要作用[4]。针对不同的应用场景，国外研究人员基于不同体系结构的高性能计算平台对其进行了移植和优化工作[5]。近年来，国内超算平台发展迅速，已显现出优越的性能。以国内某配备了 c86 处理器的超算平台（以下简称为“c86 处理器平台”）为例，该平台采用 ROCm 架构，可配置超万片异构加速芯片 DCU，算力峰值可达百亿亿次，被广泛用于生物信息、工业设计、石油勘探、动漫渲染、气象环境等计算行业。CP2K 作为一款高性能计算应用软件，源码中调用协处理器的部分基于计算统一设备架构（Compute Unified Device Architecture，CUDA）编程，仅能实现调用 CUDA 平台上的 NVIDIA GPUs 进行并行计算[6]，不能借助该平台上的海量计算资源推进相关研究和生产的进度。基于平台编译环境和异构计算设备的差异进行分析：若要实现 CP2K 软件的跨平台应用，技术上需要实现 CP2K 软件调用平台上的 DCU 加速卡、通过正确性检验等，此外，还需兼顾软件运行结果的精度，优化其运行性能[7]。

本文深入分析 CUDA 平台与国产 c86 处理器平台对于 CP2K 软件底层支持的差异，通过异构计算可移植接口（Heterogeous-compute Interface for Portability，HIP）编程标准最终实现后者对 DCU 版 CP2K 软件的底层硬件支持。其关键步骤在于：在平台上使用根据交叉编译原理生成的相应工具链依次编译 CP2K 软件运行所依赖的各种库（如 DBCSR 数学库），重新编写对应平台的 toolchain 脚本文件，然后交叉编译 CP2K 库。之后通过运行“rocm-smi”命令证实了本文所做工作可实现 CP2K 源代码调用平台上的异构加速芯片 DCU，接着选取并运行两个 benchmark 算例，验证 CP2K 软件运行的正确性，后续又基于二者采用编译级与运行级优化方法进行实验。结果证明本文采用的方法可有效提高软件的运行性能。本文的主要贡献如下：

（1）针对国产 c86 处理器平台的特殊架构，实现 CP2K 软件的移植，成功调用平台上的 DCU 加速卡。

（2）为提升 CP2K 软件的运行性能，进行编译参数优化、数学库优化、运行级优化等实验，两个测试算例各自运行前后时间的比值分别为 1.15 和 1.07。

（3）填补了国内关于 CP2K 软件面向国产硬件平台移植优化的研究空缺，为国家信创产业实现“国产化替代”做出贡献。

本文第 2 节介绍相关工作；第 3 节和第 4 节分别介绍 CP2K 软件在国产 c86 处理器平台上的移植工作，以及在此基础上进行的性能优化实验；第 5 节叙述性能对比测试；最后总结全文并展望未来。

2 相关工作

2.1 国产 c86 处理器平台

伴随国产众核处理器的发展，众核异构计算集群已成功实现商用化，并显现出优越的特性。国产 c86 处理器平台作为国产公共算力服务平台之一，基于 ROCm 架构设计，提供 rocm 等库，支持 HIP 和 OpenCL 等异构语言，兼容成熟主流的硬件和软件技术[5]。平台可配备超万片异构加速卡和超百万核心的通用计算资源，算力峰值可达百亿亿次，能够有效提高程序的运行效率，可用于人工智能和深度学习领域的相关研究[6]。其中，平台上所配置的 DCU 加速卡采用类 GPU 架构设计，精简控制单元并部署大量的计算单元，如图 1 所示。该结构特性使其在运行数据规模大、计算密集的并行程序时，突显出了优越的计算能力[8-9]。同时，DCU 通过统一的底层硬件驱动平台和层次化的软件栈，适配不同 API 接口和编译器，能够最大限度地利用已有成熟算法。

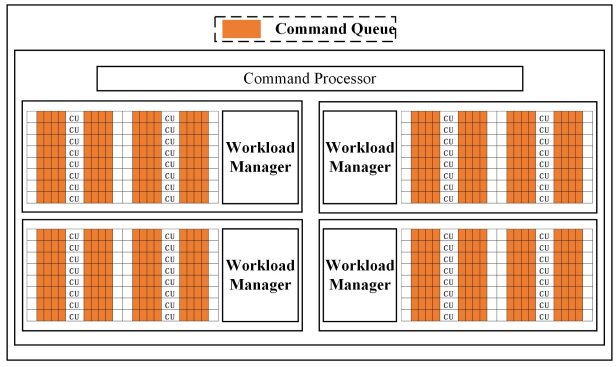

图 1 加速器硬件架构图

Fig. 1 Accelerator hardware architecture

为利用国产 c86 处理器平台海量计算资源进行相关研究，国内已有将相关程序成功移植到国产 c86 处理器平台的先例，例如，Zhang 等[7]实现了将海冰模型应用程序移植到该平台上进行相关研究，但尚无关于在国产 c86 处理器平台上移植 CP2K 源码的相关研究。为健全国产平台的软件生态，本文基于国产 c86 处理器平台移植 CP2K 软件，实现调用 DCU 加速卡，并尝试突破其运行性能。

2.2 CP2K

分子动力学模拟是一套依靠牛顿力学来模拟分子体系运动的分子模拟方法，是研究微观世界的有效手段[10-13]。它通常借助高效的软件进行模拟研究，常见的分子动力学模拟软件有 CP2K，NAMD，AMBER，GROMACS 等[8,13-16]。其中 CP2K 利用现代高性能计算体系结构，将高效算法与优秀的并行可伸缩性相结合，可研究上千个原子的大体系，已被广泛应用于模拟固体、液体、分子和生物系统等电子结构[9,17]。

例如，2017 年，Saravanan 等[18]借助 CP2K 对氧化锡电极上的 CO2 还原进行了第一性原理量子化学研究；2014 年，Brück 等[19]使用 CP2K 构建哈密顿矩阵和重叠矩阵，提出了一种基于密度泛函理论的原子量子输运模拟器；2015 年，Mones 等[20]在 CP2K 和 AMBER 两种常用的软件包中实现和验证了自适应缓冲力(AdBF)量子力学/分子力学(QM/MM)；同年，Krizek[21]使用 CP2K 代码进行全电子基组和混合泛函的基态电子结构计算，获得了优化后的开壳缺陷模型的相对稳定性等结构性质。

CP2K 作为 CUDA 平台的一个高性能计算应用软件，其源码中用于调用协处理器的部分基于 CUDA 编程模型构建，支持 MPI 和 OpenMP 技术，但仅能使用 NVIDIA GPUs 提供的并行计算引擎完成运行。为使 CP2K 软件能够在不同体系结构的高性能计算平台上运行，如基于 Intel MIC，CUDA，OpenCL 等编程标准的平台，研究人员针对不同的平台进行了移植和优化工作。比如，Reid 等[22-23]在主机上使用 GFortran 编译 CP2K，将 CP2K 代码成功移植到基于 Intel MIC 架构的 Xeon Phi Platform 上，并研究了在不进行重大代码修改或调优的情况下可以实现的性能和可伸缩性。但通过调研国内外关于 CP2K 软件在平台上的移植优化工作发现，目前国内少有工作涉及面向国产硬件平台的 CP2K 软件的移植与优化[2-3,14,24-26]。因此，若能将该软件移植到国产超算平台上，不仅可以实现 CP2K 软件在国产平台上的正常运行，还能协助相关课题的研究。

2.3 相关工具介绍

2.3.1 HIP

由于编译器和异构并行计算设备之间具有差异，原生的 CP2K 软件无法在国产 c86 处理器平台上编译安装，进而无法调用平台上的 DCU 加速卡。HIP 作为一种开源 C++运行时 API 和内核语言，提供了一系列代码自动检查与转换工具，能在 CUDA 平台上为相关软件提供接近原生的性能，以及额外的低级硬件功能。HIP 因其优越特性，目前已被广泛用于跨平台移植高性能计算应用软件。因此，HIP 可作为国产 c86 处理器平台运行时环境之上的一层很薄的封装，使 CP2K 软件忽略平台底层架构差异，在源码级别实现跨平台转码运行。其主要原理为，使用 HIP 提供的转换工具 hipify-perl 和 hipconvertinplace-perl. sh，在保持原始 CP2K 软件性能的前提下，自动检查源文件中的 CUDA API 并将其转换为相应的 HIP API。经 HIP 转换后的代码可正确链接 DCU 加速卡的相关库，以此实现连接平台的并行计算引擎。但这些转换工具对于复杂的文件包含结构会有遗漏，因此一旦编译过程出现报错，就需要多次人工调试。

2.3.2 DBCSR 数学库

CP2K 中的主要数据结构之一是矩阵。分布式压缩稀疏行(Distributed Block Compressed Sparse Row，DBCSR)数学库是进行分布式块压缩稀疏行矩阵运算的库，其能充分利用矩阵的块结构稀疏特性，从而辅助多核 CPU 和 GPU 系统有效执行稀疏矩阵-矩阵乘法等操作，以此实现高效的计算和通信。它独立于 CP2K，可通过 CUDA 和 HIP 来调用 NVIDIA GPU 和 DCU。DBCSR 数学库可以编译为 4 种主要变体，即串行版本、共享存储并行编程版本(Open Multi-Processing，OPENMP)、消息传递接口(Massage Passing Interface，MPI)版本和 OPENMP+MPI 版本。

然而 CP2K 程序文件数量多且结构复杂，移植时不仅需要转换代码文件，还需修改相应的编译构建工具。如果依次手动编译 CP2K 所需要的各个库，再在平台上编译安装 CP2K，将需要更多次的人工调试才能移植成功。为提高移植效率，本文放弃整体移植 CP2K，最终选择移植 CP2K 软件中支持核心计算任务的 DBCSR 数学库。

3 CP2K 软件至 c86 处理器平台的移植

本文借助 HIP 转换工具，以相应版本的 DBCSR 数学库为移植桥梁，最终实现在国产 c86 处理器平台上编译安装 DCU 版的 CP2K 软件，大致移植方案如图 2 所示。由于 CP2K 本身不支持 HIP，但 DBCSR 数学库有支持 HIP 的版本，因此本文在 CP2K 里重新下载编译安装支持 HIP 版本的 DBCSR 数学库。

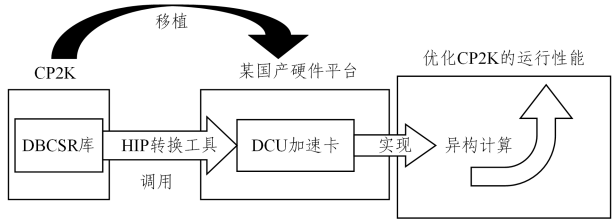

图 2 移植方案示意图

Fig. 2 Schematic diagram of migration scheme

3.1 CP2K 软件移植流程

由于原生的 CP2K 软件无法借助国产 c86 处理器平台的 DCU 加速卡进行高效的异构并行计算，因此本文提出了一种面向国产 c86 处理器平台的 CP2K 软件的移植流程，如图 3 所示。在国产 c86 处理器平台上编译安装 HIP 版的 DBCSR 库，与 CP2K 软件链接，最终实现在国产 c86 处理器平台上运行 DCU 版的 CP2K 软件。其核心在于：对 CP2K 软件中主要基于 CUDA 接口实现的 DBCSR 数学库进行代码分析，拆解对应结构体和类的封装方式，并基于 HIP 的编程标准对其重新进行实现和封装。

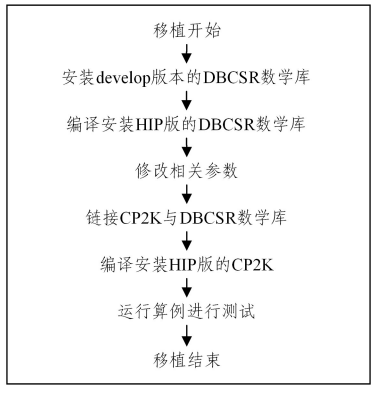

图 3 移植流程图

Fig. 3 Porting flowchart

3.2 技术实现

如果 CMake 版本过低，将导致相关软件无法编译安装，

为顺利编译 DTK 软件栈，确保 CP2K 软件移植工作顺利进行，本文对不同版本的项目构建工具 CMake 和编程语言 Python 分别进行编译安装、测试，最终选择在用户目录下以源码形式依次编译安装 CMake 3.19.6 和 Python 3.9.5。由于 DBCSR 数学库的前 3 种变体版本无法在后续优化工作中完成进程绑定，但是 OPENMP＋MPI 版本的 DBCSR 数学库支持这项操作，因此本文选择在平台上编译安装后者。具体做法为：使用 CP2K 软件中的 toolchain 脚本自动下载、编译安装 CP2K 依赖的各种库。

由于脚本链自动安装的 Python 发行版本会因版本问题影响后续编译工作正常进行，为解决这一问题，本文移除了脚本链默认加载的 Anaconda3 软件。其次，经多次实验验证，2.1.0版本的 DBCSR 库无法很好地支持 HIP，而 develop 版本的 DBCSR 库满足需求，因此本文摒弃 CP2K 里自带的 DBCSR 数学库，在 CP2K 软件中重新下载、编译安装符合要求的 develop 版本的 DBCSR 库。再者，虽然 rocm 2.9 及更高版本所提供的自动转码工具能够为转码文件自动插入 hip_run-time.h 头文件，但它会遗漏复杂的文件包含结构。

本文对比 CUDA 编程模型，依据 HIP 编程标准修改与运行 CP2K 软件相关库的参数，以此来确保经 HIP 转换后的 DBCSR 数学库的代码可以正确链接平台上的并行计算接口，核心步骤如表 1 所列。最后在平台上成功编译、安装带有 DBCSR 数学库的 CP2K 软件。

表 1　修改 Makefile 的编译选项

Table 1　Modify compilation options for Makefile

目录名称	添加
CFLAGS	-D_ACC-D_DBCSR_ACC
LIBS	-lmkl_core,-lmkl_gf_lp64 等
LDFLAGS	/public/software/compiler/rocm/rocm-3.9.1/lib 等

3.3　运行测试

为检验 DCU 版的 CP2K 软件在平台上运行的正确性、查看其对平台 DCU 加速卡的调用情况，本文选取并运行两个 benchmark 算例，即算例 H2O-256.inp 和 H2O-dft-ls.NREP2.inp（以下分别简称为“算例 1”和“算例 2”），在单节点上对软件的运行正确性进行验证操作。由于每节点里配备 4 块 DCU 加速卡，因此本文在程序运行时使用命令“salloc -N 1-p PilotCup -n 32 -gres＝dcu:4”申请 4 块 DCU 加速卡参与计算任务。如图 4 所示，某时刻该节点上的所有加速卡均成功调用，且 DCU 占用率分别为 18%，97%，54%，92%。显然程序的运行性能还有很大的优化空间，相关优化工作将在第 4 节中阐述。

```
=========================System Management Interface ===
================================================================
DCU   Temp    AvgPwr   Fan    Perf     PwrCap   VRAM%   DCU%
0     48.0c   24.0W    0.0%   manual   450.0W    97%    18%
1     48.0c   87.0W    0.0%   manual   450.0W    97%    97%
2     48.0c   27.0W    0.0%   manual   450.0W    97%    54%
3     47.0c   91.0W    0.0%   manual   450.0W    97%    92%
================================================================
================================End of SMI Log==================
```

图 4　DCU 的使用情况

Fig. 4　Usage of DCU device

4　CP2K 软件在国产 c86 处理器平台上的性能优化

本文第 3 节已实现将 CP2K 软件移植到国产某 c86 处理器平台。为充分发挥 CP2K 软件在国产 c86 处理器平台上的使用价值，使其能投入到实际应用中，本节进一步优化了 CP2K 在平台上的运行性能，依次测试并分析了几种优化方法对其性能的优化效果。如图 5 所示，程序优化方法按高性能计算机的层次结构，从上至下可分为：算法级与代码级的程序优化、编译级与运行级的程序优化、硬件级的程序优化。其中，算法级与代码级程序优化针对具体代码或算法进行操作，而本文工作着重于优化软件的整体运行性能，故而摒弃了该层次的优化方法；又因硬件级程序优化的实质在于以增加硬件投入为代价来提升程序的性能，所以这种优化方法虽然效果明显，但因完成难度高且成本高，本文不予采用。

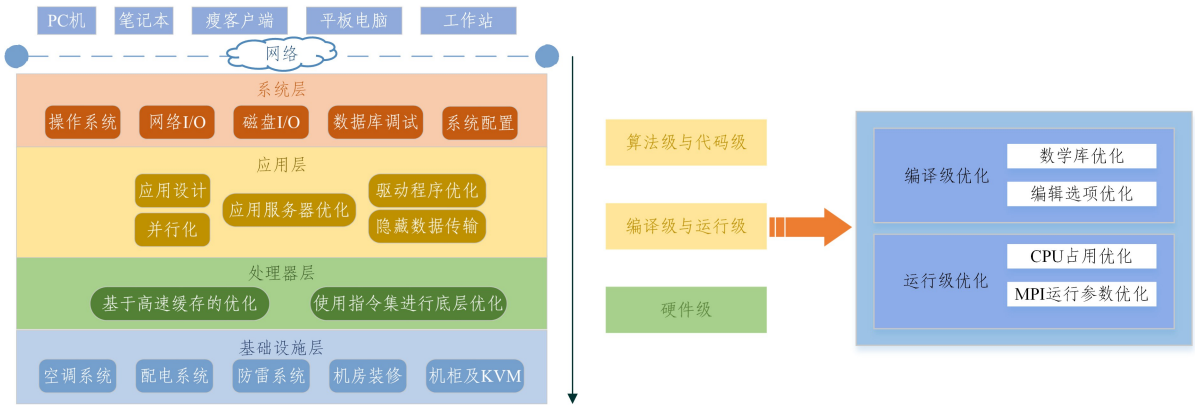

图 5　高性能计算机的层次结构及优化方法

Fig. 5　Hierarchical structure and optimization method of high performance computer

本文衡量了技术成本及难度后，最终选择在编译级与运行级展开优化工作。其中，编译级的程序优化包括编译选项优化和数学库优化，运行级的程序优化包括 CPU 占用优化和 MPI 运行参数优化。这两部分内容将分别在 4.1 节和 4.2 节中详细阐述。

4.1　编译级程序优化

编译级程序优化指通过编译器、链接器生成性能较好的二进制程序文件，从而优化程序的性能。因此，本文依次进行了编译选项的优化、选择性能较好的外部库（数学库）进行优化。

4.1.1 编译选项优化

目前,CP2K 软件支持 GNU 编译器和 Intel 编译器进行编译操作,前者具备的多种编译参数能提供不同级别的代码优化,详细参数如表 2 所列。

表 2 各种编译参数及作用

Table 2 Various compilation parameters and their functions

编译参数	作用
-O1	提供基础级别的优化
-O2	提供更加高级别的代码优化
-O3	提供最高级别的代码优化
-Os	优化代码尺寸
-messes3	针对 CPU 指令级进行优化

编程语言译器(GNU Compiler Collection,GCC)是由 GNU 编译器套件开发的,本文选用 GCC 编译 CP2K 软件代码。经测试发现,编译选项优化应适度。优化参数越多,优化参数级别越高,程序的编译时间就越长,运行效率就越低,还会影响程序结果数值的精度[25,27]。因此本文在选择编译选项进行实验时,综合考虑了整个程序的运行效率和程序运行结果精度。

4.1.2 数学库优化

数学库指完成常用数学计算操作的外部库,在 CP2K 软件中合理调用数学库能在一定程度上优化程序。CP2K 支持的数学库有 Intel 的 MKL 库、LIBXSMM 库、SCALAPACK 库、LIBXC 库和 rocBLAS 库。

FFTW(Fast Fourier Transform in the West)库是一套进行离散傅里叶变换(Discrete Fourier Transform,DFT)计算的数学库。如表 3 所列,本文使用的 MKL 库可以实现 FFTW API。但它会与由 CP2K 脚本自动安装的另一版本的 FFTW 库发生冲突,影响最终计算结果的精度。为提升算例运行结果的精度,本文只保留前者进行实验。后续将尝试在算例中加入 rocBLAS 数学库进行实验测试,以期加快算例运行,进而优化 CP2K 软件的运行性能。

表 3 几种数学库及作用

Table 3 Several mathematical libraries and their roles

数学库名称	作用
MKL	完成傅里叶变换操作
LIBXSMM	完成专门化的稠密和稀疏矩阵操作
SCALAPACK	完成若干线性代数求解功能操作
LIBXC	完成交换相关函数操作
rocBLAS	向量和矩阵运行的规范

4.2 运行级优化

运行级优化指在二进制文件生成后,运行时进行的优化方法。本文进行了其中的 CPU 占用优化和 MPI 运行参数优化。

4.2.1 CPU 占用优化

并行化指将一个算法或程序由串行执行修改为并行执行。Amdahl 定律用于描述求解固定规模问题时,程序使用并行化方法可达到的最大加速比。其计算式如式(1)所示:

$$S=(WS+WP)/[WS+(WP)/p]=1/(1/p+f\times(1-1/p)) \tag{1}$$

其中,WS 为程序串行部分的执行时间,WP 为程序可并行化部分的执行时间,p 为并行进程数,f 为程序串行部分执行时间占总时间的比例,且 $f=WS/(WS+WP)$。当 $p\to\infty$ 时,$S=1/f$,即对于固定规模的问题,并行系统所能达到的加速上限为 $1/f$。根据 Amdahl 定律可知,随着使用的 CPU 数量增加,程序性能逐渐提高,达到一个极值。当总 CPU 数量超过一个阈值后,伴随 CPU 数量的增加,操作系统资源调度会产生更多的时间开销,程序的性能会逐渐降低。

为量化探究程序运行性能与其运行时 CPU 所占用核心数之间的关系,本文在实验中分别令"OMP_NUM_THREADS"的数值为 1,2,4,8,然后改变单节点的 CPU 占用核心数,测试描绘出不同 CPU 占用核心数-程序性能曲线。

4.2.2 MPI 运行参数优化

MPI 库可为超算应用提供高效的通信服务,目前已有 OpenMPI,Intelmpi,MPICH 等多种实现版本。为适应不同的系统平台,满足不同应用程序的需求,开发人员在设计 MPI 实现方式时设置了成百上千的参数[28-29],如表 4 所列。在程序运行时选用合适的 MPI 运行参数,可以优化程序的运行性能。

表 4 MPI 运行参数

Table 4 MPI operating parameters

MPI 的实现版本名	执行进程绑定的命令行参数
Openmpi	-bind-to core
Intelmpi	-env I_MPI_PIN_DOMAIN
MPICH	binding cpu/cpu:sockets/cpu:cores
MVAPICH	MV2_CPU_BINDING_POLICY

进程绑定作为 MPI 运行参数优化的方式之一,通过限制 MPI 进程使用的 CPU 核心数,来避免操作系统因频繁调度而出现自身开销过大、性能降低的问题。为使操作系统负载均衡,本文基于平台上的单节点(每个节点配置 1 颗 32 核心 2.5GHz 主频的 c86 处理器,且物理核心数为 16)采用进程绑定进行优化操作。但由于 DBCSR 模块在 MPI 进程间运行 Cannon 算法,而该算法为减少分块矩阵乘法运算的存储量将矩阵均匀分布在每个进程中,所以该算法要求进程数为完全平方数(如 1,4,9,16 或 25)。又由于在进行进程绑定操作时,其要求进程数目应为物理核心数 的整数倍,因此本文在该优化实验中指定 16 个进程,且每个进程利用两个 CPU 核心。

5 性能测试对比

5.1 性能的评估指标

为量化分析程序运行性能的优化效果,本文选用算例运行时间和其优化后获得的加速比作为评估指标。其中,程序性能指单位时间内程序所能完成的任务的数量或完成特定任务所需时间的倒数。但由于任务类型千差万别,难以对比,故通常使用程序运行时间的倒数作为程序性能的度量。为更准确地度量程序性能的优化程度,本文引入加速比(Speedup)这一概念,其计算方法为优化后的程序性能与优化前的程序性能的比值。定义方式如式(2)所示:

$$Speedup=\frac{PerfOpt}{PerfOri} \tag{2}$$

其中,$PerfOpt$ 指优化后的程序性能,$PerfOri$ 表示优化前的

程序性能。比值越大，代表性能优化效果越好。

5.2 测试平台与测试算例

本文完成了 CP2K 软件在国产 c86 处理器平台上的移植与优化工作。5.3 小节将基于国产 c86 处理器平台，分别运行算例 1 和算例 2 进行测试实验，以此量化分析第 4 节中几种方法的优化效果。测试平台的配置信息如表 5 所列，实验中均调用节点上的 4 张 DCU 加速卡参与计算任务。

表 5 国产 c86 处理器平台配置信息

Table 5 Domestic c86 processor platform configuration

软硬件	版本及型号
C/Fortran 编译器(gcc/g++)	7.3.1
HIP 编译器 hipcc	2.8.19361
openmpi	4.0.2
c86 处理器	32 核心 2.5GHz 主频
DCU 加速卡(每节点 4 块)	HBM2 片上高速内存
DDR4 2666 ECC REG 内存/GB	16

5.3 实验结果与分析

5.3.1 编译选项优化实验

本文在默认编译选项的基础上，分别使用-O1，-O2，-O3，-O2-msse3，-O2 -fif-conversion，-O2-finlinefunctions 这 6 种编译选项来分别测试算例 1 和算例 2。为减小运行时间的误差，本文采用连续运行 3 次取平均值的方式进行测量。

各编译选项的运行时间如图 6 所示，当使用单个编译选项时，通过对比发现，使用-O2 编译选项时两个算例的平均运行时间最短，分别为 1 343.032 s 和 81.933 s。为进一步优化，后续工作以只使用-O2 编译选项运行算例的时间为基准，分别增加-msse3，-fif-conversion，-finline-functions 依次进行实验。结果发现：-O2 选项和-fif-conversion 选项搭配使用时算例 1 的平均运行时间最短，为 1 298.121 s；-O2 选项和-msse3 选项搭配使用时算例 2 的平均运行时间最短，为 81.332 s。

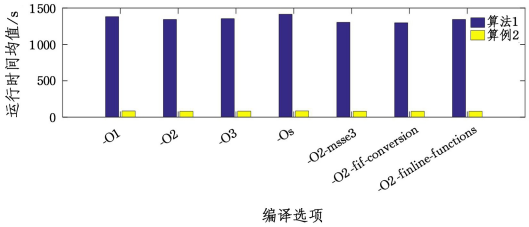

图 6 各编译选项的运行时间均值

Fig. 6 Average run time of each compile option

实验结果表明，在原先默认编译选项基础上搭配使用-O2 和-fif-conversion 选项时算例运行时间最短，运行效率最高。因此，本文在原先默认编译选项的基础上改用-O2，并添加-fif-conversion，以期尽可能地缩短程序运行时间。

5.3.2 数学库优化实验

本文在更改两个算例的迭代次数后发现，算例 2 的运算结果精度不高。为解决这一问题，本文通过对比官方版本算例输出日志，去掉了 CP2Ktoolchain 脚本安装的 FFTW 库。测试后发现计算结果精度提升至小数点后 7 位。

在数学库优化方面，还加入了加速器卡函数 rocBLAS 库进行实验，测试结果如表 6 所列。以两个算例各自优化前的运行时间为基准，实验结果表明，去掉脚本自动安装的 FFTW 库并加入 rocBLAS 库后，加速器库函数 rocBLAS 对两个算例的优化效果不明显。其中，算例 1 和算例 2 优化前后运行时间的比值分别为 1.15 和 1.07。

表 6 数学库优化实验结果

Table 6 Math library optimization experimental results

算例	优化前时间/s	优化后时间/s	比值
算例 1	1 500	1 300	1.15
算例 2	88	82	1.07

5.3.3 CPU 占用优化实验

CPU 占用优化实验中以算例 2 的运行时间为基准，分别设置 1，2，4，8 线程进行对比实验。该优化方法的核心在于：通过改变程序运行时使用的 CPU 核心数来探索最优的程序性能。

实验结果如图 7 所示，当线程数设置为 1，CPU 核心数增加至 10 时，算例 2 的运行时间减少得最明显，为 831.27 s。但在线程数为 8 的情况下，由于频繁的线程切换操作对程序性能造成了更大的负面影响，因此该条件下，随着 CPU 核心数的增加，算例 2 的运行时间呈上升趋势。结合 Amdahl 定律及测试结果发现，随着 CPU 核心数增加，算例 2 的运行时间逐渐降低至一个极值，当所使用的 CPU 核心数增加到 10 以后，运行时间不再随着核心数的增多而有明显变化，由此验证了 CPU 占用优化方法的有效性。

此外，经分析得出：CPU 核心数为平方数(如 16 或 25)时，整体性能较好。其中，当 CPU 核心数为 25 时，算例 2 的运行时间最短，为 83.199 s，即最优 CPU 占用的核心数为 25。

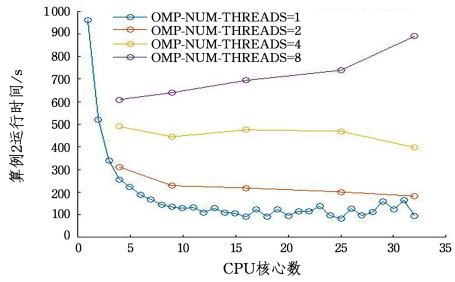

图 7 算例 2 测试结果图

Fig. 7 Test result graph of example 2

5.3.4 MPI 运行参数优化实验

在 MPI 运行参数优化实验中，本文采用 CPU 进程绑定方式。具体操作为：在程序运行中指定 16 进程 32 核心。为检验进程绑定对程序性能的优化效果，本文结合 CPU 占用优化实验中最优占用的实验结果，即以采用 25 进程 1 线程运行方式运行的时间为基准，与本实验所得的运行时间结果做对比。实验结果如表 7 所列。

表 7 运行时间对比

Table 7 Runtime comparison

运行方式	算例 1 时间/s	算例 2 时间/s
25 进程 1 线程	1 366.298	82.313
16 进程 32 核心	1 106.895	69.858

分析表 7 可知，与算例 2 相比，进程绑定可大幅提升两个算例的运行速度。其中，算例 1 的运行时间减少 259.403 s，算例 2 的运行时间减少 12.455 s。由此验证了 MPI 运行参数

优化方法的可行性，提升了 CP2K 软件的运行性能。

结束语 本文基于国产 c86 处理器平台，对国外开源分子动力学软件 CP2K 进行移植并优化。为解决原生 CP2K 软件不能调用 DCU 加速卡的问题，本文选择编译安装 HIP 版的 DBCSR 库，将其正确链接到 CP2K 中，从而在部分移植 CP2K 软件的情况下实现调用 DCU 加速卡。在性能测试及优化方面，选取两个 benchmark 算例，即 H20-256.inp 算例和 H2O-dftls.NREP2.inp 算例，以它们的运行时间以及优化前后各自运行时间的比值为基准，分别采用编译选项优化、数学库优化、CPU 占用优化和 MPI 运行参数优化这 4 种优化方法来测试优化效果。通过实验发现，使用-O2 编译选项优化可略微提升算例的运行速度，加入 rocBLAS 库和使用运行级优化对算例 1 的优化效果较算例 2 更明显。进行编译选项优化时，-O2 选项和-fif-conversion 选项搭配使用时的算例运行时间最短，运行效率最高。数学库优化的测试结果表明，去掉脚本自动安装的 FFTW 库可提升计算结果的精度，加入 rocBLAS 库可缩短两个算例的运行时间。其中算例 1 缩短了 200 s 以上，算例 2 缩短了近 6 s，且两者优化前后运行时间的比值分别为 1.15 和 1.07。运行级优化的实验结果表明，随着使用 CPU 的核心数增加，程序的性能逐渐提高，达到一个极值。但当核心数增加到 10 后，算例 2 的运行时间不再有明显变化。此外还发现 CPU 核心数为平方数时整体性能较好；进程绑定可大幅度提升算例 1 和算例 2 的运行速度。

为进一步探索两个测试算例运行性能的优化方向，本文在河南师范大学超算集群上编译安装 CUDA 版 CP2K，调用 8 张 3090 GPU 卡，运行两个测试算例。在未进行任何优化的情况下，算例 1 的运行时间为 571.813 s(本文中为 1106.895 s)，算例 2 的运行时间为 35.219 s(本文中为 69.858 s)。可见，HIP 版的 CP2K 软件的运行性能还有很大的优化空间。

为探索优化其性能的有效方法，本文分析 DBCSR 库的代码发现，其中的 jit_kernel 函数采用了 JIT 编译技术。插入时间测试函数，分别运行两个算例后发现：该函数在程序运行中均被调用 16 次，平均每次耗时分别为 2.9 s 和 2.3 s，总共耗时为 46.4 s 和 36.8 s，在算例 1 中占比为 4.2%，在算例 2 中占比为 52.5%。由此合理猜想，若能将其转为 AOT 运行前编译方式，将大幅提升算例 2 的运行性能。此外，CP2K 软件运行的核心操作是进行大量的浮点数运算，采用加速卡设备(如 DCU)能够有效提升其计算效率，但这也意味着在主机端(Host)和设备端(Device)之间增加了大量的数据拷贝操作。因此，为提高程序的并行效率[30-31]，下一步工作的重点是减少进程间通信时间以及主机端与设备端间的数据拷贝时间。

参 考 文 献

[1] TIAN Z, CHEN Y F. Performance Optimization of Molecular Dynamics Simulation on the Light of Sunway Taihu Lake[J]. Journal of Software, 2021, 32(9): 2945-2962.

[2] YAO W J. Kamui· Implementation and Optimization of Molecular Dynamics Software on Taihu Light[D]. Hefei: University of Science and Technology of China, 2017.

[3] MENG D L, WEN M H, WEI J W, et al. Porting and Optimization of OpenFOAM on the Light of Sunway Taihu Lake[J]. Computer Science, 2017, 44(10): 64-70.

[4] VANDEVONDELE J, BORSTNIK U, HUTTER J. Linear Scaling Self-consistent Field Calculations with Millions of Atoms in the Condensed Phase[J]. Journal of Chemical Theory and Computation, 2012, 8(10): 3565-3573.

[5] SU N Y. Analysis on the Development Model of China's Supercomputing Technology Catch-up and Overtake[J]. Journal of National University of Defense Technology, 2021, 43(3): 86-97.

[6] YAN B H, AN H, LIANG W H, et al. Implementation and Optimization of Discrete Execution Model for I/O Intensive Applications[J]. Journal of Chinese Computer Systems, 2019, 40(12): 2619-2623.

[7] ZHANG Y C X, FENG L B, LIANG J G. Research on Parallel Performance Tuning of Sea Ice Model Using Sugon Supercomputer[J]. Software Guide, 2021, 20(6): 80-85.

[8] KÜHNE T D, IANNUZZI M, DEL BEN M, et al. Cp2k: An Electronic Structure and Molecular Dynamics Software Package-quickstep: Efficient and Accurate Electronic Structure alculations[J]. The Journal of Chemical Physics, 2020, 152(19): 194103.

[9] MONDAL A, GAULTOIS M W, PELL A J, et al. Large-scale Computation of Nuclear Magnetic Resonance Shifts for Paramagnetic Solids Using CP2K[J]. Journal of Chemical Theory and Computation, 2018, 14(1): 377-394.

[10] QI X F, ZHANG X H, LI J Z, et al. Molecular Dynamics Simulation Study of NC/NG Blended System[J]. Journal of Ordnance Industry, 2013, 34(1): 93-99.

[11] WEN Y H, ZHU R Z, ZHOU F X, et al. Main Techniques for Molecular Dynamics Simulation[J]. Progress in Mechanics, 2003(1): 65-73.

[12] WU N, CUI D D, JI B C, et al. Design of Innovative Experimental Project in Undergraduate Enzyme Engineering Based on GROMACS Software* —Molecular Dynamics Simulation of Microbial Glutaminase in Aqueous Solution[J]. Chemical Education, 2021, 42(20): 102-107.

[13] YAO W J, CHEN J S, SU Z C, et al. Migration and Optimization of NAMDSoftware Based on Sunway Taihu Light[J]. Computer Engineering and Science, 2017, 39(6): 1022-1030.

[14] PENG L, CHEN J S, AN H. AMBER Software Migration and Optimization Based on Sunway Taihu Light[J]. Computer Engineering, 2020, 46(12): 12-20.

[15] STONE J E, HYNNINEN A P, PHILLIPS J C, et al. Early Experiences Porting the Namd and Vmd Molecular Simulation and Analysis Software to Gpu-Accelerated Openpower platforms [C]// International Conference on High Performance Computing. 2016: 188-206.

[16] NIE N, CHANGJUN H, YUNQUAN Z, et al. Comparison of Large-scale Molecular Dynamics Software for Materials Microstucture Evolution[J]. Journal of Frontiers of Computer Science & Technology, 2017, 11(3): 355.

[17] HUTTER J, IANNUZZI M, SCHIFFMANN F, et al. CP2K: Atomistic Simulations of Condensed Matter Systems[J]. Wiley Interdisciplinary Reviews: Computational Molecular Science,

2014，4(1)：15-25.

[18] SARAVANAN K，BASDOGAN Y，DEAN J，et al. Computational investigation of CO2 electroreduction on tin oxide and predictions of Ti，V，Nb and Zr dopants for improved catalysis [J]. Journal of Materials Chemistry A，2017，5(23)：11756-11763.

[19] BRÜCK S，CALDERARA M，BANI-HASHEMIAN M H，et al. Towards Ab-initio Simulations of Nanowire Field-effect Transistors[C]// 2014 International Workshop on Computational Electronics(IWCE). IEEE，2014：1-3.

[20] MONES L，JONES A，GÖTZ A W，et al. TheAdaptive Buffered Force QM/MM Method in the CP2K and Amber Software Packages[J]. Journal of computational chemistry，2015，36(9)：633-648.

[21] KRIZEK E A R. Computational Study of Radiation-induced Defects in Topaz[D]. London：University College London，2015.

[22] BETHUNE I，REID F，LAZZARO A. CP2K Performance from Cray XT3 to XC30 [J/OL]. https://cug. org/proceedings/cug2014_proceedings/includes/files/pap127-file2. pdf.

[23] REID F，BETHUNE I. Evaluating CP2K on Exascale Hardware：Intel XeonPhi[J/OL]. https://hgpu. org/?p=12283.

[24] ZHANG Y Z，CAO W D，BU J D，et al. Migration and Optimization of GROMACS 2020 on ROCm Platform[J]. Computer Engineering and Science，2021，43(11)：1901.

[25] SUN G H，HU X，WANG R. OpenCV Porting and Optimization Based on FT-M7002[C]//Proceedings of the 22nd Annual Conference on Computer Engineering and Technology and the 8th Microprocessor Technology Forum. 2018.

[26] WANG Y C，LIN X H，CAI L J，et al. Using OpenACC to Port and Optimize GTC-P on Taihu Light[J]. Journal of Computer Research and Development，2018，55(4)：875-884.

[27] CAI J. Structural Analysis and Back-end Porting Practice of GCC Compilation System[D]. Hangzhou：Zhejiang University，2004.

[28] ZHENG W X. Research on MPI Runtime Parameter Tuning Method and Key Technology in Parallel Computer System[D]. Changsha：National University of Defense Technology，2019.

[29] XIE S W，HUA B. Communication Optimization of MPI/RDMA Library in Virtual Machine Environment[J]. Journal of Chinese Computer Systems，2021，42(7)：1505-1510.

[30] CHAI X Q，DONG Y L，LI J F. Profit-oriented Task Scheduling Algorithm in Hadoop Cluster[J]. EURASIP Journal on Embedded Systems，2016，2016 (1)：1-8.

[31] CHAI X Q，DONG Y L，LI J F. An Algorithm for Improved Similarity and Collaborative Filtering in Social Networks [J]. IJSSST (International Journal of Simulation：Systems，Science and Technology)，2016，17(22)：12. 1-12. 6.

FAN Lilin，born in 1970，Ph.D，associate professor，master tutor. His main research interests include intelligent business and intelligent transportation.

CHAI Xuqing，born in 1982，senior engineer，master tutor. Her main research interests include high-performance computing and machine learning.

（责任编辑：何杨）