软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(8):2307−2318 [doi: 10.13328/j.cnki.jos.006003]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

复杂异构计算系统 HPL 的优化*

黎雷生[1,2], 杨文浩[1], 马文静[1,2], 张 娅[1,2], 赵 慧[1], 赵海涛[1,2], 李会元[1,2], 孙家昶[1,2]

[1](中国科学院 软件研究所 并行软件与计算科学实验室,北京 100190)

[2](计算机科学国家重点实验室(中国科学院 软件研究所),北京 100190)

通讯作者: 黎雷生, E-mail: leisheng@iscas.ac.cn

摘 要: 当今世界的主流超级计算机越来越多地使用带有加速器的异构系统.随着加速器的浮点性能不断提高,超级计算机内计算节点的 CPU、内存、总线、网络以及系统架构都要与之相适应.HPL(high performance Linpack)是高性能计算机评测的传统基准测试程序,复杂异构系统给 HPL 评测带来很多机遇与挑战.针对带有 GPU 的异构超级计算机系统,提出一套新的 CPU 与加速器计算任务分配方式,提出平衡点理论指导 HPL 性能优化.为了优化 HPL 程序,提出了使用 CPU 与加速器协同工作的 look-ahead 算法和行交换连续流水算法,实现了加速器、CPU、网络等部件的高度并行.此外,为带有加速器的系统设计了新的 panel 分解和行交换的实现方法,提高了加速器的利用率.在每个节点带有 4 个 GPU 的系统上,单节点 HPL 效率达到了 79.51%.

关键词: 复杂异构系统;平衡点理论;panel 分解加速;连续流水线算法

中图法分类号: TP303

中文引用格式: 黎雷生,杨文浩,马文静,张娅,赵慧,赵海涛,李会元,孙家昶.复杂异构计算系统 HPL 的优化.软件学报,2021,32(8):2307−2318. http://www.jos.org.cn/1000-9825/6003.htm

英文引用格式: Li LS, Yang WH, Ma WJ, Zhang Y, Zhao H, Zhao HT, Li HY, Sun JC. Optimization of HPL on complex heterogeneous computing system. Ruan Jian Xue Bao/Journal of Software, 2021,32(8):2307−2318 (in Chinese). http://www.jos.org.cn/1000-9825/6003.htm

Optimization of HPL on Complex Heterogeneous Computing System

LI Lei-Sheng[1,2], YANG Wen-Hao[1], MA Wen-Jing[1,2], ZHANG Ya[1,2], ZHAO Hui[1], ZHAO Hai-Tao[1,2], LI Hui-Yuan[1,2], SUN Jia-Chang[1,2]

[1](Laboratory of Parallel Software and Computational Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

[2](State Key Laboratory of Computer Science (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

Abstract: Nowadays, the mainstream supercomputers in the world adopt heterogeneous systems with accelerators more and more. The increase of float point computation performance of the accelerators requires other components to match its speed, including CPU, memory, bus, and network. High performance Linpack (HPL) is the traditional benchmark for high performance computers. Complex heterogeneous systems have brought both opportunities and challenges to the benchmarking with HPL. Therefore, for heterogeneous supercomputers, a new task partitioning scheme between the CPU and the accelerators is proposed, using the balance point theory to guide the optimization of HPL. For optimizing HPL, a look-ahead algorithm is proposed to coordinate the collaboration of CPU and the

* 基金项目: 中国科学院战略性先导科技专项(C 类)(XDC01030200); 国家重点研发计划(2018YFB0204404, 2016YFB0200601); 国家自然科学基金(11871455, 11971016)

Foundation item: Strategic Priority Research Program of the Chinese Academy of Sciences (Category C) (XDC01030200); National Key Research and Development Program of China (2018YFB0204404, 2016YFB0200601); National Natural Science Foundation of China (11871455, 11971016)

本文由“国产复杂异构高性能数值软件的研制与测试”专题特约编辑孙家昶研究员、李会元研究员推荐.

收稿时间: 2019-08-20; 修改时间: 2019-12-05; 定稿时间: 2020-01-22

accelerators, as well as a contiguous row-swap algorithm, enabling the parallelism among CPU, accelerators, and network. Besides, new panel factorization and row-swap implementations have been designed for the system with accelerators, improving the effectiveness and efficiency of the usage of accelerators. With the configuration of 4 GPUs on each computing node, HPL efficiency of 79.51% on a single node.

Key words: complex heterogeneous system; balance point theory; panel factorization acceleration; contiguous row-swap algorithm

HPL(high performance Linpack)是评测计算系统性能的程序,是早期 Linpack 评测程序的并行版本,支持大规模并行超级计算系统[1],其报告的每秒浮点运算次数(floating-point operations per second,简称 FLOPS)是世界超级计算机 Top500 列表排名的依据.

基于 Linpack 的 Top500 排名开始于 1993 年,每年发表两次.2007 年 11 月及之前的列表,排名前 10 位的超级计算机的计算能力全部由同构 CPU 提供.2008 年 6 月 Top500 首台性能超过 1 PFLOPS 的超计算机 Roadrunner 使用异构 CPU 结构,通用 CPU 执行调度、管理等任务,加速 CPU 主要负责计算密集任务.2009 年 11 月排名第 5 位的 Tianhe-1 使用了 CPU+GPU 的异构架构.此后榜单上排名前 10 的系统 CPU+加速器的架构成为趋势.2019 年 6 月最新的排名显示,前 10 位中有 7 台系统使用 CPU+加速器架构,其中使用 GPU 加速器的 5 台,使用 XEON Phi 的 1 台,使用 Matrix-2000 的 1 台[2].

HPL 浮点计算集中在 BLAS 函数,特别是 DGEMM.对于同构 CPU 架构,优化 BLAS 函数特别是 DGEMM 性能是提高 HPL 的浮点效率的关键.Dongarra 等人[1]总结了 2002 年之前 Linpack 发展历史和 BLAS 函数的优化方法.

对于 CPU+加速器架构,优化方向集中于 CPU 端 BLAS 函数、加速器端 BLAS 函数、CPU 与加速器之间负载分配和数据传输等.Kurzak 等人[3]优化多核 CPU 多 GPU 的 HPL,其 4 个 GPU 的 DGEMM 浮点性能达到 1 200 GFLOPS.Bach 等人[4]面向 AMD CPU 与 Cypress GPU 架构优化了 DGEMM 和 HPL,HPL 效率达到 70%.Wang 等人[5]采用自适应负载分配方法优化 CPU 与 GPU 混合系统 HPL 性能,并调优 HPL 参数,目标系统 Tianhe1A 浮点性能达到 0.563 PFLOPS.Heinecke 等人[6]基于 Intel Xeon Phi Knights Corner 优化了 DGEMM,使用动态调度和改进的 look-ahead 优化 HPL,100 节点集群 HPL 效率达到 76%.Gan 等人[7]优化基于加速器的 DGEMM,并应用于 Tianhe-2 HPL 评测.

随着以 GPU 为代表的加速器技术的发展,加速器浮点性能越来越高,CPU 与加速器的浮点性能差距越来越大.2019 年 6 月,Top500 排名第一的 Summit 系统[8]的 1 个节点装备 2 个 CPU,理论浮点性能 1 TFLOPS,装备 6 个 GPU,理论浮点性能 42 TFLOPS.本文研究的目标系统使用 CPU+GPU 异构架构,每个节点装备 1 个 32 核 CPU,4 个 GPU,CPU 浮点计算性能约是 GPU 的 1/61.同时加速器本身的结构也变得越来越复杂,通过增加特定的硬件满足特定领域的需求,如 Nvidia GPU 的 Tensor Core 等.已有研究使用 Tensor Core 的强大的半精度运算能力混合双精度计算开发了 HPL-AI[9],报告 Summit 的 HPL-AI 性能是全双精度 HPL 的 2.9 倍.并且已有应用采用混合精度算法加速计算,从 HPL 和应用角度来看,混合精度都是值得研究的方向.面对这种新的计算架构,内存、总线、网络以及系统设计都要与之相适应,形成复杂的异构计算系统,这为 HPL 评测带来很多机遇与挑战.

本文在基础 HPL 代码之上,针对目标系统实现 HPL 并研究其优化方法.第 1 节简要介绍基础算法.第 2 节介绍目标系统基本情况.第 3 节描述复杂异构系统 HPL 平衡点理论.第 4 节介绍复杂异构系统 HPL 高效并行算法.第 5 节介绍基础模块的性能优化,包括 panel 分解优化和行交换 long 算法的优化.第 6 节介绍目标系统 HPL 实验和结果分析.第 7 节总结并展望未来的工作.

1 基础算法

HPL 算法使用 64 位浮点精度矩阵行偏主元 LU 分解加回代求解线性系统.矩阵是稠密实矩阵,矩阵单元由伪随机数生成器产生,符合正态分布.

线性系统定义为:

$$Ax=b;\ A\in R^{N\times N};\ x,b\in R^{N}.$$

行偏主元 LU 分解 $N\times(N+1)$系数矩阵$[A,b]$:

$$P_r[A,b]=[[L\cdot U],y];\ P_r,L,U\in R^{N\times N};\ y\in R^N.$$

其中,P_r 表示行主元交换矩阵,分解过程中下三角矩阵因子 L 已经作用于 b,解 x 通过求解上三角矩阵系统得到:

$$Ux=y$$

HPL 中 LU 分解浮点运算次数 $\frac{2}{3}N^3-\frac{1}{2}N^2$,回代浮点运算次数 $2N^2$.

HPL 采用分块 LU 算法,每个分块是一个 NB 列的细长矩阵,称为 panel.LU 分解主循环采用 right-looking 算法,单步循环计算 panel 的 LU 分解和更新剩余矩阵.基本算法如图 1 所示,其中 $A_{1,1}$ 和 $A_{2,1}$ 表示 panel 数据.需要特别说明的是,图示矩阵是行主顺序,HPL 代码中矩阵是列主存储的.

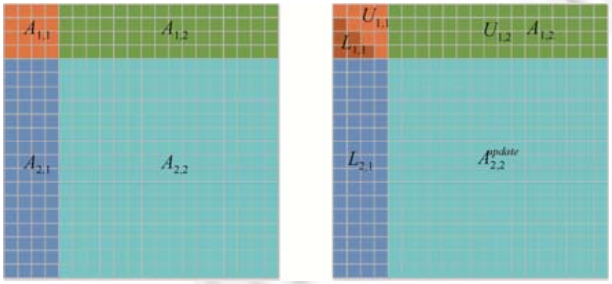

Fig.1 Block LU algorithm

图 1 分块 LU 算法

计算公式如下:

$$\left[\frac{L_{1,1}}{L_{2,1}},U_{1,1}\right]=LU\left(\frac{A_{1,1}}{A_{2,1}}\right),$$

$$U_{1,2}=L_{1,1}^{-1}A_{1,2},$$

$$A_{2,2}^{update}=A_{2,2}-L_{2,1}U_{1,2}.$$

第 1 个公式表示 panel 的 LU 分解,第 2 个公式表示求解 U,一般使用 DTRSM 函数,第 3 个公式表示矩阵更新,一般使用 DGEMM 函数.

对于分布式内存计算系统,HPL 并行计算模式基于 MPI,每个进程是基本计算单元.进程组织成二维网格.矩阵 A 被划分为 $NB\times NB$ 的逻辑块,以 Block-Cycle 方式分配到二维进程网格,数据布局示例如图 2 所示.

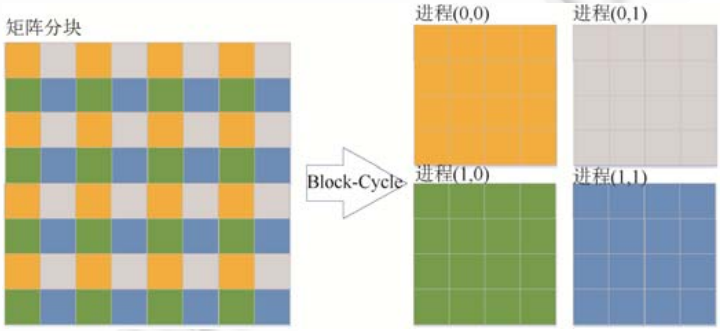

Fig.2 Block-Cycle algorithm

图 2 Block-Cycle 算法

对于具有多列的进程网格,单步循环只有一列进程执行 panel 分解计算,panel 分解过程中每一列执行一次

panel 的行交换算法选择并通信最大主元行.Panel 分解计算完成后,把已分解数据广播到其他进程列.HPL 基础代码包含 6 类广播算法,可以通过测试选择较优的算法.

HPL 采用行主元算法,单步矩阵更新之前,要把 panel 分解时选出的最大主元行交换到 U 矩阵中,需要执行未更新矩阵的主元行交换和广播.主元行交换和广播后,每个进程获得完整的主元行数据.

矩阵更新包括两部分计算,一是使用 DTRSM 求解 U,二是使用 DGEMM 更新矩阵数据.

LU 分解完成后,HPL 使用回代求解 x,并验证解的正确性.

2 系统介绍

2.1 系统硬件

优化 HPL 的目标系统是典型的复杂异构系统,计算部件由通用 CPU 和计算加速器 GPU 组成,网络接口和互联系统是高速 Infiniband 网络.

2.2 基础软件

目标 CPU 兼容 x86 指令集.编译器使用 gcc,MPI 使用 OpenMPI.

HPL 基础代码使用 2.3 版本.HPL 会调用多个 BLAS 函数.CPU 端 BLAS 使用针对目标 CPU 优化的 OpenBLIS,支持 OpenMP 实现多核并行.GPU 端 BLAS 使用针对目标 GPU 优化的 BLAS.

3 复杂异构系统 HPL 平衡点理论

从硬件的角度来看,CPU 的每个 Die 与对应的内存系统、GPU 和网络接口组成相对独立的系统.因此,在选择并行方案时,采用 1 个进程使用 1 个 Die,并且与相应的内存、GPU 和网络接口绑定的架构.HPL 基础代码中只有 1 列或 1 行进程时执行流程不同,较大规模并行一般是多行多列进程,因此使用 1 个节点 4 进程作为研究对象,覆盖多行多列进程执行流程.

对于复杂异构系统 HPL 计算,需要协同调度 CPU、GPU、PCIe 和通信网络等资源.首先研究 CPU 与 GPU 计算任务分配方式,进程内 CPU 与 GPU 以及进程间的数据传输,并且建立线程模型控制 CPU、GPU 和网络接口协同计算.然后分析 HPL 各部分的性能,提出复杂异构系统的平衡点理论指导性能优化.

3.1 计算任务分配

文献[4,6]等研究的异构系统中,CPU 执行 panel 分解计算,加速器执行矩阵更新的 DTRSM 和 DGEMM 计算.CPU 计算能力可以达到加速器计算能力的 10%或更高,除了 panel 分解,CPU 还可以计算部分矩阵更新.并且由于加速器内存限制,矩阵不能完全存储在加速器,矩阵更新一般采用流水的方式.

经过分析和实验对比,由于 panel 分解控制流程复杂,并且大量调用 GPU 计算效率较低的 DGEMV、DSCAL 等函数,不适合在 GPU 端实现,因此,仍然采用 CPU 计算 panel 分解的方案.

对于矩阵更新来说,目标系统 CPU 和 GPU 之间的数据传输成为瓶颈,导致矩阵更新流水线方法不再适用.针对单个 GPU 分析,假设矩阵规模 N=58368(占用约 80%的系统内存),NB=384,矩阵更新的 DGEMM 效率为 85%.可以计算出矩阵更新计算时间 0.52s.PCIe 单向传输矩阵数据的理论时间 1.70s.数据传输时间超过 GPU 的计算时间,导致 GPU 超过 2/3 处于空闲状态,GPU 计算能力不能充分利用.

因此,采用矩阵数据常驻 GPU 内存的方式,矩阵更新时避免传输整个剩余矩阵.这种模式下,矩阵的规模受限于 GPU 设备内存的容量.为了运算方便,CPU 端内存保留 GPU 端数据存储的镜像.

3.2 数据传输

数据传输包括进程内主机内存与 GPU 设备内存之间数据传输和进程间数据传输.主机内存与 GPU 设备内存数据传输使用 PCIe 总线,一般采用 DMA 异步模式,数据传输的同时 CPU 和 GPU 可以执行计算任务.进程间数据传输使用 MPI.如果数据传输的进程位于同一个节点,MPI 使用共享内存等技术提高数据传输性能.节点之

间的数据传输通过网络接口.

由于矩阵数据常驻 GPU 内存,panel 分解前把当前的 panel 数据从 GPU 传输到 CPU.Panel 分解在 CPU 端执行 panel 的主元选取与广播,使用基础代码的算法.已完成 panel 分解的数据,进程内部通过 PCIe 传输到 GPU.进程之间通过 MPI 传输,非当前 panel 分解进程接收到 panel 数据后,再通过 PCIe 传输到 GPU.

相比于基础代码,矩阵更新行交换和广播增加了 CPU 与 GPU 数据传输过程.矩阵的行在 GPU 内存不连续,数据传输前后需要执行封装和重排.数据传输关系如图 3 所示.

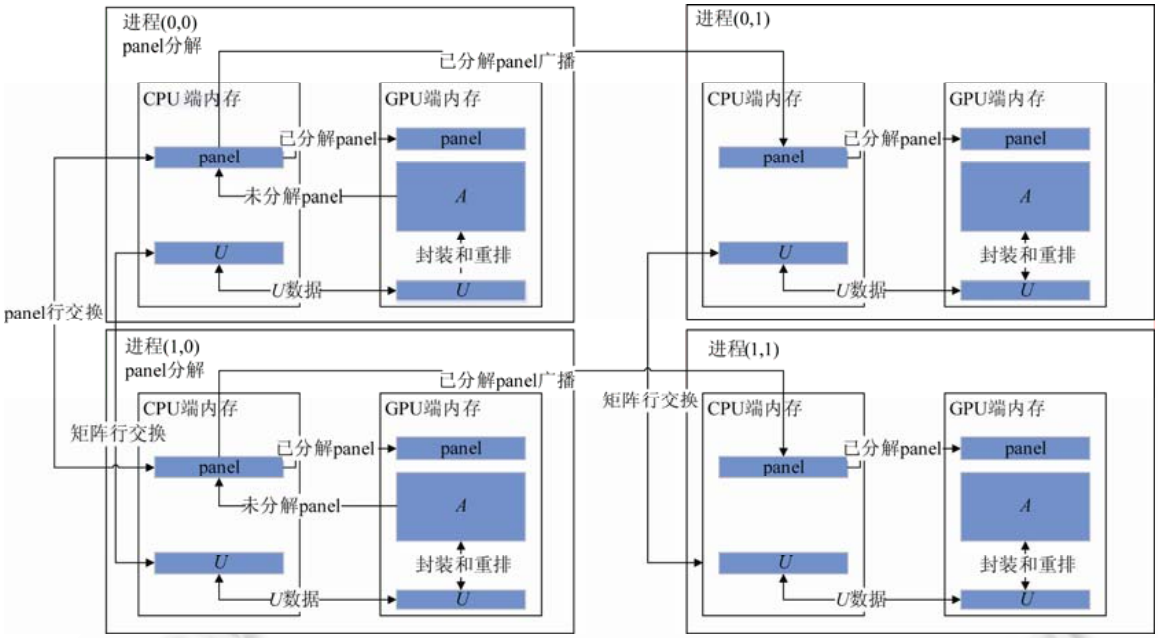

Fig.3 Data transmisson

图 3 数据传输

3.3 多线程模型

在 1 个节点 4 个进程的架构下,每个进程使用 8 个 CPU 核心,使用多线程管理和调度这些核心.Panel 分解和 panel 广播共享数据,执行顺序存在依赖关系,放在同一个线程.矩阵行交换和矩阵更新共享数据,执行顺序存在依赖关系,放在同一个线程.两组操作之间相对独立,分别绑定到特定核心.两个线程之间通过信号量同步.

Panel 分解线程调用和管理 OpenBLIS 函数,OpenBLIS 使用 OpenMP 线程模型.Panel 分解调用 OpenBLIS 函数时,与 panel 分解的其他过程没有资源共享和冲突,OpenBLIS 的 OpenMP 线程绑定的核心可以包括 panel 分解主线程的核心,也就是最多可绑定 7 个核心.

3.4 复杂异构系统协同计算平衡点理论

目标系统是复杂异构系统,涉及 CPU、GPU、PCIe 和网络接口等部件的协同计算.为了更好分析和评测各部分对 HPL 性能的影响,指导性能优化,提出平衡点理论.

HPL 算法中,矩阵更新的浮点计算占据绝大部分,这部分计算由 GPU 完成.性能分析过程中,矩阵更新被认为是有效计算,其他功能模块包括 panel 分解和矩阵行交换的目标是为矩阵更新提供数据准备.为了获得较高的效率,要尽可能保证 GPU 处于工作状态,发挥 GPU 浮点计算能力.对于 panel 分解和矩阵行交换,尽可能让它们与 GPU 计算并行,使用 GPU 计算隐藏 panel 分解和矩阵行交换.

研究没有使用特殊优化、单步循环各部分时间以及串行执行的总时间,如图 4 所示.采用 2×2 进程布局,图

中是进程 0 的时间曲线.进程 0 在计算过程中处于不同的位置,包括当前列-当前行和非当前行-非当前列两种情况,图中"A"代表当前列-当前行,"B"代表非当前行-非当前列.两种情况下执行的计算有差异,每种运算有两条时间曲线.panel 分解时间在不同规模下稍有波动.可以看出,单步循环串行执行时间远大于 GPU 执行有效计算的时间.那么首先必须做好矩阵更新与 panel 分解和行交换的并行,GPU 计算时间隐藏 panel 分解与和行交换时间.

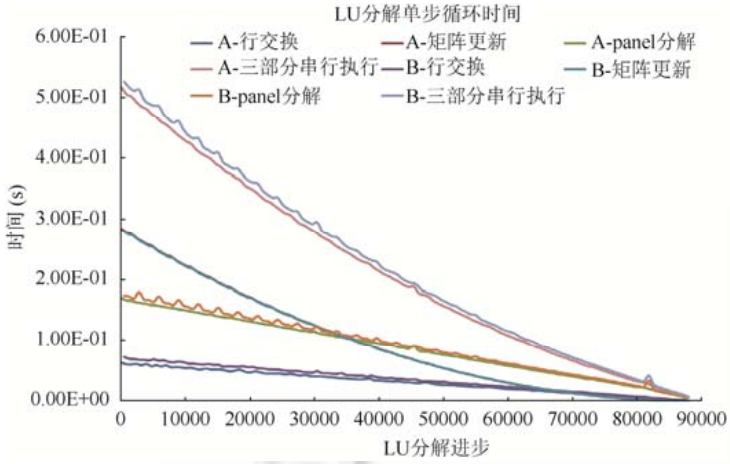

Fig.4 Time of panel factorization, row swap, matrix update and each loop

图 4 Panel 分解、行交换、矩阵更新和三部分串行执行累加时间

另一方面,HPL 的 LU 分解过程分为两个阶段,矩阵更新时间大于 panel 分解和行交换的时间的阶段和矩阵更新时间小于 panel 分解直至小于行交换时间的阶段.在 panel 分解与矩阵行交换并行的情况下,两个阶段的分界点即矩阵更新与 panel 分解时间曲线的交点就是 CPU 与 GPU 协同计算的平衡点.

针对 1 节点 4 进程进行分析,N=88320 情况下,从计算规模来看,平衡点在 37.82%,LU 计算量占 76.00%,之后 GPU 不能充分发挥性能.有必要继续优化 panel 分解和行交换,减少执行时间,平衡点后移,延长 GPU 高效率计算时间,进而提高整体的计算效率.平衡点理论可以很好地指导 HPL 的优化工作.

4 复杂异构系统 HPL 高效并行算法

根据第 3.4 节的分析,首先做好 GPU 计算与 panel 分解和行交换的并行算法,实现 GPU 计算重叠其他部分的时间.

4.1 GPU参与计算的look-ahead算法

基础代码用 look-ahead 算法实现 panel 分解与 panel 广播重叠执行.计算过程中保存两个 panel 的数据结构.执行当前 panel 分解的进程列,优先执行当前 panel 列数据的行交换和矩阵更新,然后执行 panel 分解计算并发起 panel 广播,广播数据传输的同时更新剩余矩阵.对于当前不计算 panel 分解的进程列,执行上一次循环的行交换和矩阵更新,行交换和矩阵更新计算过程中监测本次 panel 广播数据通信,数据到达后,执行广播通信流程.

由于加入了 GPU 计算资源,look-ahead 算法相比于基础版本有所改变.按照第 3.3 节的线程模型,在执行当前 panel 分解计算的进程中,panel 分解线程 CPU 优先执行 panel 列的行交换,然后 GPU 优先更新 panel 分解的列.CPU 等待 panel 数据更新后,发起 panel 数据从 GPU 内存到 CPU 内存的传输,传输结束后执行 panel 分解计算,最后发起 panel 广播.矩阵更新线程在 panel 分解列行交换完成后发起剩余矩阵的行交换,然后调用 GPU 矩阵更新计算.对于非当前 panel 分解计算进程,矩阵更新线程执行行交换和调用 GPU 矩阵更新.Panel 分解线程只负责处理 panel 广播的操作.

4.2 行交换连续流水线算法

矩阵行交换的前提是已接收 panel 分解计算的行交换索引数据(包含在 panel 广播数据中),并且上一次迭代矩阵更新已完成.HPL 基础代码中,行交换完成后启动矩阵更新计算,如果采用这种算法,GPU 在行交换执行的过程是空闲的.文献[6]提出行交换流水线算法优化行交换流程.在使用 GPU 的环境,也可以采用这一算法.使用行交换流水线算法,对矩阵更新的行分段,首先执行第 1 段的行交换,然后 GPU 执行第 1 段的矩阵更新,同时执行第 2 段的行交换,依次类推.采用这种算法,GPU 计算与行交换数据传输重叠执行,减少了 GPU 的空闲时间.但是,单步循环执行第 1 段行交换的这段时间内 GPU 是空闲的.

为了避免 GPU 等待,提出了连续流水线算法.算法中,第 1 个分段完成上一次循环矩阵更新和当前进程接收到下一次循环的行交换信息之后,不需要等待上一次循环矩阵更新全部完成,就执行第 1 分段的下一次循环的行交换.采用这种算法,下一次循环第 1 分段的行交换被上一次循环矩阵更新隐藏,下一次循环第 1 分段的矩阵更新隐藏第 2 分段的行交换,后续分段继续流水.

使用连续流水算法避免单步循环的流水线启动过程,如果接收到行交换信息早于矩阵更新,则 GPU 可以一直处于工作状态.从第 3.4 节可以看出,行交换信息在平衡点之前是早于矩阵更新的,而这一阶段正是浮点运算量较大的情况,在这一阶段充分发挥 GPU 计算能力,可以提高整体的计算效率.

行交换分段方法选择倍数递增,也就是第 2 段列数是第 1 段列数的倍数,依次类推,增长因子可调. 连续行交换流水算法下实现 look-ahead 算法,对于当前 panel 分解,增加 panel 分解数据提前行交换和矩阵更新的分段.

5 基础模块性能优化

从平衡点理论来看,平衡点之前的部分 GPU 计算时间可以隐藏 panel 分解和行交换时间.随着计算的推进,panel 分解和行交换成为系统性能的瓶颈.为了进一步提高效率,需要针对 panel 分解和行交换进行优化.

5.1 Panel分解优化

5.1.1 基本参数调优

Panel 分解浮点计算集中在 BLAS 函数,使用针对目标系统优化的 Hygon OpenBLIS 库.Panel 分解使用递归算法,中间递归层次的浮点运算集中在 BLAS 的 DTRSM 和 DGEMM 两个函数.当递归层次包含的列数小于等于阈值时,使用非递归算法,浮点运算集中在 BLAS 的 DGEMV、DTRSV、DSCAL、IDAMAX 等函数.对于每一列的 LU 分解,需要通过交换和广播通信主元所在的 panel 行,并记录主元行交换信息.交换和广播通信使用 binary-exchange 算法.

Panel 分解计算有关参数有 *NB*、*NBMIN*、*PFACT*、*RFACT*、*DIV* 等.*NB* 取决于 GPU 执行矩阵更新的效率,同时考虑 CPU 与 GPU 计算的平衡.当节点规模较小时,*NB*=384;当节点规模较大时,*NB*=256.通过参数调优,选择优化的参数组合,见表 1.

Table 1 Parameters of panel factorization

表 1 Panel 分解参数

参数	值
NB	384 或 256
NBMIN	8
PFACT	Left
RFACT	Left
DIV	2

进一步分析 panel 分解各部分时间,0 号进程主要计算函数时间见表 2.

Table 2 Time of BLAS functions in panel factorization

表 2 Panel 分解 BLAS 函数时间

函数	时间(s)
DGEMM	5.26+00
DTRSM	7.14−02
DSCAL	1.92−01
DGEMV	2.87−01
DTRSV	5.72−03
DIDAMAX	2.56−01

5.1.2 GPU 加速 panel 分解 DGEMM

从第 5.1.1 节可以看出,panel 分解中,DGEMM 时间占有最大比例,需要进一步优化.DGEMM 在 panel 分解递归层次调用.根据 panel 分解的 left-looking 算法,从左到右执行 LU 分解,左侧的 subpanel 完成分解之后,执行 DGEMM,更新相同层次的右侧的 subpanel,GPU 可以加速这部分 DGEMM.

GPU 加速 DGEMM 的算法过程,首先把左侧 subpanel 和相应的 *U* 数据传输到 GPU 设备内存,然后执行 DGEMM 更新 GPU 内存的右侧 subpanel,更新后数据传输到 CPU 端内存,继续执行后续的 panel 分解.算法增加了 CPU 和 GPU 之间的数据传输,但由于 GPU 执行 DGEMM 的速度远高于 CPU,在一定规模 DGEMM 的情况下,这一过程总的执行时间少于 CPU 端执行 DGEMM 时间,整体性能提高.

并不是所有的 panel 分解 DGEMM 都可以使用 GPU 加速,如果传输的时间开销大于 DGEMM 加速的效果,就不能采用这种方式.实际计算中,使用参数控制采用 GPU 加速 DGEMM 的阈值,通过调优获得最佳的性能.

5.1.3 Panel 广播优化

5.1.3.1 避免数据封装

基础代码中,panel 分解数据使用 CPU 端矩阵 *A* 的内存,*A* 是列存储,panel 分解的列是分段连续的.使用 MPI 传输 panel 数据之前,需要执行数据封装,把数据复制到发送缓冲区,而复制操作会带来一定的时间开销.

为此提出一种避免数据封装的方法.待分解的 panel 数据是从 GPU 内存复制的,通过使用二维复制接口把 panel 数据复制到连续存储区域.Panel 分解计算完成后,MPI 接口直接使用缓冲区数据,避免了数据封装.

5.1.3.2 Panel 广播流水

基础代码中,panel 分解完成后才会执行 panel 广播,计算与数据传输串行,并且一次性数据传输较大,传输时间较长.通过观察可以看出,left-looking 算法中,左侧 subpanel 计算完成后,主体部分不再变化,后续只有发生行交换的数据变化.

因此,提出一种广播流水算法,对已分解的 subpanel 数据提前发起广播,这种情况下,panel 数据广播与后续 panel 分解计算并行.采用广播流水算法,panel 计算完成后,只需要传输最后的 subpanel 和行交换发生变化的数据,相比于传输整个 panel 数据,缩短了传输时间.

广播流水算法与 GPU 加速 panel 分解 DGEMM 协同使用,subpanel 的数据广播与 panel 加速 DGEMM 过程中的 CPU 与 GPU 之间数据传输、GPU 执行 DGEMM 计算并行,充分利用系统的 CPU、GPU、PCIe 和网络接口资源.

HPL 包含的 6 种广播算法都可以使用广播流水优化.

第 5.1.2 节描述的 GPU 加速 panel 分解 DGEMM 和 panel 广播流程紧密耦合,表 3 给出了优化前后伪代码对比.

Table 3 Pseudocode of accelerated panel DGEMM and broadcast pipeline

表 3 Panel DGEMM 加速和广播流水伪代码

Panel 分解基础伪代码	Panel 分解 DGEMM 加速和广播流水伪代码

HPL_pdrpanllN()
{
if (subpanel 列数小于等于 *NBMIN*)
{执行非递归计算;返回;}
do
{
CPU 执行 DTRSM 计算;
CPU 执行 DGEMM 计算;
递归调用 HPL_pdrpanllN();
}while (subpanel 计算没有完成);
}
	DHPL_pdrpanllN()
	{
	if (subpanel 列数小于等于 *NBMIN*)
	{执行非递归计算;返回;}
	do
	{
	if (subpanel 列数大于 GPU 加速列数)
	{
	CPU 传输数据到 GPU;
	GPU 异步执行 DTRSM 和 DGEMM 计算;
	if (待广播的列数大于广播流水阈值)
	{
	CPU 广播已分解的前一个 subpanel;
	}
	CPU 同步 GPU;
	GPU 传输数据到 CPU;
	} else {
	CPU 执行 DTRSM 计算;
	CPU 执行 DGEMM 计算;
	}
	递归调用 DHPL_pdrpanllN();
	if(全部 panel 计算完成){
	CPU 广播最后的 subpanel 和变化的数据;
	}
	} while (subpanel 计算没有完成);
	}

5.2 行交换long算法优化

矩阵更新行交换有两种算法可选,bianary-exchange 算法和 long 算法,经过分析和评测选择 long 算法[10].第 4.2 节从行交换与 GPU 并行计算的角度针对行交换流程做了优化,本节优化具体的行交换算法.

Long 算法包括两个步骤,spread 和 roll.基础代码中,首先是 spread 过程,当前行进程将需要换出的行分发给其他所有进程.然后,非当前行进程把接收的数据与本地需要换出的数据交换.接着是 roll 过程,总共需要进行 P–1 次传输.每次传输所有的进程都要参与,每个进程只与自己的邻居交换.第 1 步,先把自己所拥有的一部分与左(右)邻居交换,然后,把自己刚刚得到的数据与右(左)邻居交换.这样左右交替,直到所有的数据都交换完毕,每个进程都拥有了全部的 U.最后所有进程还需要把所有的数据进行重排,放置到矩阵 A 相应的位置.

基础代码中,spread 和 roll 使用同一块数据缓冲区,为了避免数据冲突,spread 和 roll 是严格串行的.通过观察发现,spread 的接收数据与 roll 的发送数据并没有依赖关系,spread 和 roll 算法上是可并行性.基于以上观察与分析,将 spread 的接收缓冲与 roll 的发送缓冲分离.这样,非当前进程行在等待接收 spread 数据时,就可以执行 roll 数据封装,并把数据传输到发送缓冲区,这部分时间就可以被隐藏.

在使用上述的缓冲区分离算法之后,spread 和 roll 已经没有依赖关系,因此提出了一种新的行交换方法,即交换 spread 和 roll 的执行顺序,如图 5(b)所示.对当前行进程来说,在 roll 操作之前,只需将本地需要交换的数据拷贝到发送缓冲区即可,也就是减少了行交换的启动时间.在 roll 执行网络传输的同时,当前行进程可以将 spread 所需的数据封装,并异步传输到发送缓冲区.roll 执行完成后,开启执行 spread.同时,把 roll 过程接收的数据异步传输到 GPU,并执行数据交换.在 spread 过程结束后,非当前行进程再将换入的数据传输到 GPU,并交换到相应位置,当前行进程对需要进行内部交换的数据执行 GPU 上的本地交换.这种算法可以实现网络通信传输、GPU 数据封装和数据交换计算以及 CPU 与 GPU 数据传输相互重叠,减少行交换执行时间.

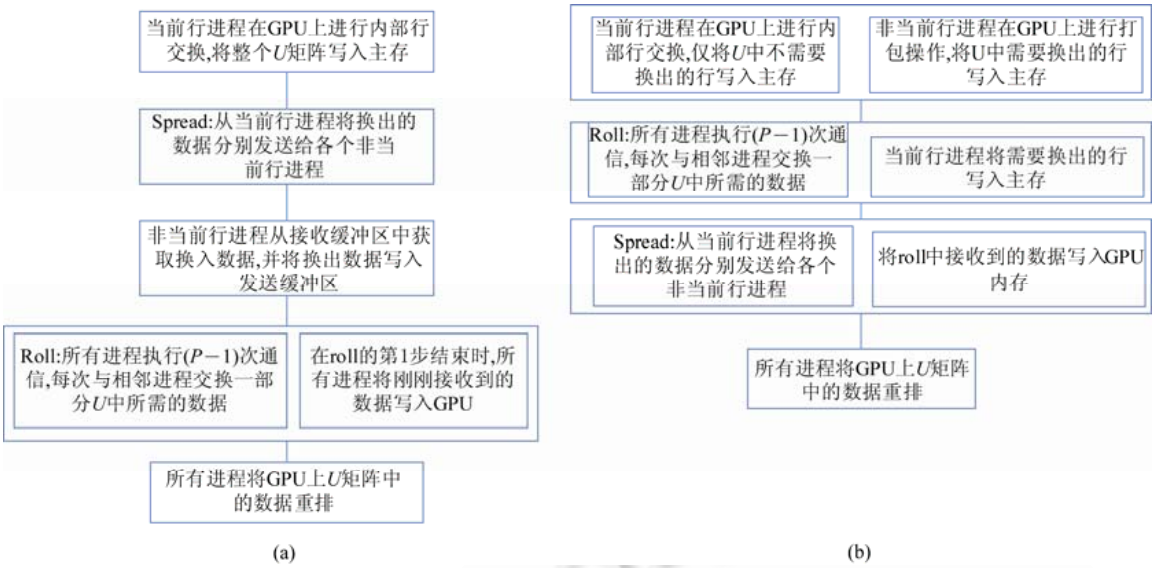

Fig.5 Original row swap algorithm (a), and modified row swap procedure (b)

图 5 原始行交换算法(a)及新的行交换算法(b)

6 实验与分析

6.1 单节点性能分析

首先针对单节点实验,分析评估各种优化方法的效果.根据平衡点理论,优化的目标是 GPU 尽可能地处于高效工作状态,执行有效的计算,也即 GPU 尽可能地连续执行矩阵更新.图 6 对比了优化后单步循环的时间和 GPU 矩阵更新时间.图 6 给出了 1 个节点内 0 号进程的时间曲线."A"表示当前行-当前列,"B"表示非当前行-非当前列.可以看出,N=88320 情况下,LU 分解矩阵规模 56.09%之前单步循环时间接近 GPU 矩阵更新时间.56.09%之后,当前进程执行 panel 分解,单步循环时间仍然接近 GPU 矩阵更新时间,GPU 大部分时间处于工作状态,说明算法优化已经达到较好的效果.对于非当前 panel 分解进程,增加了等待 panel 广播数据的时间开销,GPU 有一段时间处于空闲状态.

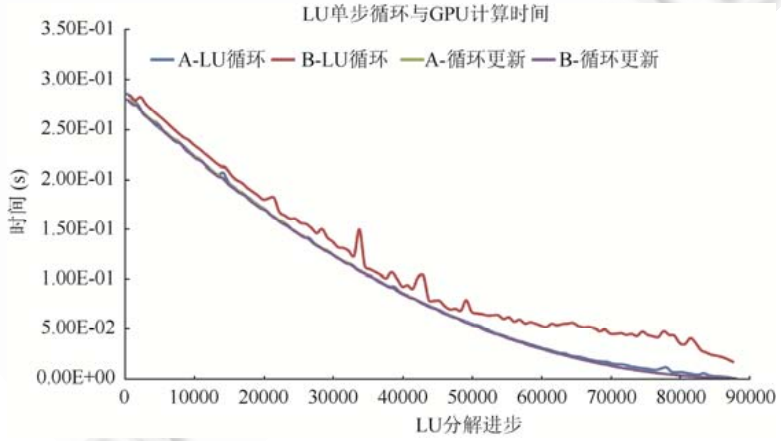

Fig.6 GPU time vs. total time of each loop

图 6 GPU 时间与单步循环总时间

可以看出,图 6 优化版本的平衡点是 56.09%,图 4 未优化版本版本是 37.82%.从 LU 分解的计算量来看,前 56.09%占总的 LU 分解计算量的 91.53%,大部分时间充分发挥了 GPU 的计算能力.优化后单节点 HPL 效率达到了 79.51%.

和优化有关的参数主要是 GPU 加速 DGEMM 和 panel 广播流水的层次,这里都是 3.行交换非当前列第 1 段列数是待更新列数的 1/8,增长因子是 2.Panel 分解参数和表 1 一致.

6.2 可扩展分析

多节点 HPL 较大规模可扩展,目前最大规模 4 096 个节点.HPL 较大规模并行有关的参数和效率见表 4,表中列出了针对相应节点数执行的实验中效率最高的配置.

Table 4 Parameters and efficiency of multi-nodes HPL

表 4 多节点 HPL 参数和效率

节点数	*N*	*NB*	*P*	*Q*	SWAP	BCAST	HPL 效率(%)
1	88 320	384	2	2	优化 long	流水 1ring	79.51
4	176 640	384	4	4	优化 long	流水 2ringM	80.01
16	353 280	384	8	8	优化 long	流水 2ringM	77.69
64	706 560	384	16	16	优化 long	流水 2ringM	74.89
256	1 376 256	384	32	32	优化 long	流水 2ringM	73.72
1 024	2 752 512	384	64	64	优化 long	流水 2ringM	72.00
1 936	3 784 704	384	88	88	优化 long	流水 2ringM	70.87
4 096	5 505 024	256	64	64	优化 long	流水 2ringM	67.50

参数 *N* 是较大规模节点 HPL 评测中比较重要的参数,每个节点计算的矩阵列数一般是 *NB* 的倍数,多节点的 *N* 正比于节点数平方根.*N* 值越大,计算效率较高的矩阵更新占比越大,一般来说 HPL 效率越高,但 *N* 受限于 GPU 的内存容量,同时还要考虑系统的负载.*NB* 使用 384 或 256.*P* 和 *Q* 确定二维进程网格的组织方式,经过验证,*P*=*Q* 时目标系统性能最优.Panel 广播和矩阵行交换分别使用优化的 long 算法和广播流水算法.

从 HPL 效率来看,4 节点效率高于单节点效率,原因在于 4 节点矩阵规模大于单节点,效率较高的计算占比较大,并且可以抵消 4 节点规模下网络传输路径增加带来的性能损失.随着规模的扩大,网络传输对性能的影响增大,HPL 效率逐步降低,但降低的趋势比较平缓.

7 结论与未来的工作

针对复杂异构系统,分析 HPL 算法的特点,提出 CPU 与 GPU 协同计算方法,提出平衡点理论指导各方面优化.实现 CPU 与 GPU 协同计算的 look-ahead、行交换连续流水线算法隐藏 panel 分解和行交换时间,并优化 panel 分解和行交换算法,延长 GPU 计算占优的时间,最终提高整个系统的 HPL 效率,单节点效率 79.51%.

当前的超级计算机正向百亿亿级迈进,目标系统采用的节点内通用 CPU 装备加速器,配合高速的节点内总线被认为是较有潜力的一种架构.未来工作主要是针对百亿亿级计算架构,开展加速器 DGEMM 优化、CPU 与加速器协同计算和混合精度等相关研究.同时,百亿亿级超级计算机节点规模继续增加,节点数量有可能达到数万甚至更多,节点间的互连也更加重要.需要进一步研究 HPL 大规模可扩展能力,评测行交换算法和广播算法的性能,并进行相应的优化.

References:

[1] Dongarra J J, Luszczek P, Petitet A. The LINPACK Benchmark: Past, present and future. Concurrency and Computation: Practice and Experience, 2003,15(9):803−820.

[2] Official website. 2020. http://www.top500.org

[3] Kurzak J, Luszczek P, Faverge M, Jack Dongarra. Programming the LU factorization for a multicore system with accelerators. In: Proc. of the Int'l Conf. on High Performance Computing for Computational Science. Berlin, Heidelberg, 2013. 28−35.

[4] Bach M, Kretz M, Lindenstruth V, Rohr D. Optimized HPL for AMD GPU and multi-core CPU usage. Computer Science—Research and Development, 2011,26:153–164.

[5] Wang F, Yang CQ, Du YF, Chen J, Yi HZ, Xu WX. Optimizing Linpack benchmark on GPU-accelerated petascale supercomputer. Journal of Computer Science and Technology, 2011,26(5):854–865.

[6] Heinecke A, Vaidyanathan K, Smelyanskiy M, Kobotov A, Dubtsov R, Henry G, Shet A, Chrysos G,Dubey G. Design and implementation of the Linpack benchmark for single and multi-node systems based on Intel Xeon Phi coprocessor. In: Proc. of the IEEE 27th Int'l Symp. on Parallel and Distributed Processing. Los Alamitos, 2013. 126–137.

[7] Gan XB, Hu YK, Liu J, Chi LH, Xu H, Gong CY, Li SG, Yan YH. Customizing the HPL for China accelerator. Science China Information Sciences, 2018,61(4):Article No.042102.

[8] Offical website. 2020. https://www.olcf.ornl.gov/summit/

[9] Offical website. 2020. https://www.hpcwire.com/2019/06/19/summit-achieves-445-petaflops-on-new-hpl-ai-benchmark/

[10] Offical website. 2020. http://www.netlib.org/benchmark/hpl/algorithm.html

黎雷生(1981－),男,博士,副研究员,主要研究领域为并行计算.

赵慧(1984－),女,博士,助理研究员,主要研究领域为高性能计算.

杨文浩(1993－),男,博士生,助理工程师,主要研究领域为高性能计算,数值计算方法.

赵海涛(1981－),男,博士,副研究员,CCF 专业会员,主要研究领域为高性能工程,科学计算.

马文静(1981－),女,博士,副研究员,CCF 专业会员,主要研究领域为高性能计算.

李会元(1973－),男,博士,研究员,博士生导师,主要研究领域为高性能计算,计算数学.

张娅(1984－),女,博士,副研究员,主要研究领域为计算数学,并行计算.

孙家昶(1942－),男,研究员,博士生导师,主要研究领域为科学与工程计算的方法、理论与应用,并行计算.