软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(8):2319−2328 [doi: 10.13328/j.cnki.jos.006004]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

国产异构系统上 HPL 的优化与分析*

水超洋[1,2], 于献智[1,2], 王银山[1,2], 谭光明[1,2]

[1](中国科学院 计算技术研究所,北京 100190)

[2](中国科学院大学,北京 100190)

通讯作者: 谭光明, E-mail: tgm@ict.ac.cn

摘 要: 随着异构系统成为建造超级计算机的重要选择,如何让 CPU 与加速器协调工作以充分发挥异构系统的计算性能具有重要意义.HPL 是高性能计算领域最重要的基准测试程序,传统面向纯 CPU 系统的 HPL 算法通过加速器加速矩阵乘法的做法已经无法取得很好的性能.针对这一问题,提出了基于国产处理器-国产加速器异构系统的 HPL 性能模型和多线程细粒度流水 HPL 算法.完成了一个轻量级跨平台异构加速框架 HPCX,以实现跨平台的 HPL 算法.该性能模型能够准确地预测类似异构系统的 HPL 性能.该 HPL 算法在 NVIDIA GPU 平台上性能超过了 NVIDIA 官方闭源 nvhpl 程序 9%.在国产处理器-国产加速器平台 512 个节点的规模上,优化的 HPL 算法实现了 2.3 PFLOPS 实测峰值性能和 71.1%的浮点效率.

关键词: HPL;异构系统;跨平台;性能建模;E 级计算

中图法分类号: TP303

中文引用格式: 水超洋,于献智,王银山,谭光明.国产异构系统上 HPL 的优化与分析.软件学报,2021,32(8):2319−2328. http://www.jos.org.cn/1000-9825/6004.htm

英文引用格式: Shui CY, Yu XZ, Wang YS, Tan GM. Optimization and analysis of HPL on domestic heterogeneous system. Ruan Jian Xue Bao/Journal of Software, 2021,32(8):2319−2328 (in Chinese). http://www.jos.org.cn/1000-9825/6004.htm

Optimization and Analysis of HPL on Domestic Heterogeneous System

SHUI Chao-Yang[1,2], YU Xian-Zhi[1,2], WANG Yin-Shan[1,2], TAN Guang-Ming[1,2]

[1](Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijing 100190, China)

Abstract: As heterogeneous system becomes one of the most important choices to build super computers, how to orchestrate CPU and accelerator to leverage the great computability of heterogeneous systems is of great significance. HPL is the most important benchmark in HPC field, traditional HPL algorithm targeting at CPU-only systems cannot achieve high performance by only offloading matrix multiplication workload to accelerators. To solve this problem, this work proposes a HPL performance model and a multithread fine-grained pipelining algorithm for domestic-processor-domestic-accelerator heterogeneous system. Meanwhile, a light weight cross-platform heterogeneous framework is implemented to carry out a cross-platform HPL algorithm. The proposed performance model predicts HPL performance accurately on similar heterogeneous systems. On NVIDIA platform, the proposed HPL algorithm outperforms the NVIDIA proprietary counterparts by 9%. On domestic-processor-domestic-accelerator platform, the finally optimized Linpack

* 基金项目: 国家重点研发计划(2018YFB0204400, 2016YFB0201305, 2016YFB0200803, 2016YFB0200300); 中国科学院战略性先导科技专项(C 类)(XDC01030000); 国家自然科学基金(61972377, 61432018, 61702483); 中国科学院前沿科学重点研究计划(QYZDJ-SSW-JSC035)

Foundation item: National Key Research and Development Program of China (2018YFB0204400, 2016YFB0201305, 2016YFB0200803, 2016YFB0200300); Strategic Priority Research Program of the Chinese Academy of Sciences (Category C) (XDC01030000); National Natural Science Foundation of China (61972377, 61432018, 61702483); Key Research Program of Frontier Sciences of the Chinese Academy of Sciences (QYZDJ-SSW-JSC035)

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: 2019-08-16; 修改时间: 2019-12-05; 定稿时间: 2020-01-22

© 中国科学院软件研究所 http://www.jos.org.cn

program achieves 2.3 PFLOPS on 512 nodes, with floating-point efficiency 71.1%.

Key words: HPL; heterogeneous system; cross-platform; performance modeling; exascale computing

HPL(high performance linpack)是目前HPC领域中最重要的基准测试程序之一,其性能被用作TOP500的排名依据[1,2].HPL 可以从超级计算机的计算性能、访存性能、网络性能、可用性和稳定性等各个方面给予超级计算机综合评价.HPL 的优化具有重要意义,在学界长期受到关注和研究.HPL 的优化可以为其他科学计算应用的并行优化提供有价值的参考.

自1993年TOP500[3]排名开始,榜单中的超级计算机的体系结构一直在发生变化.图1展示了排名第一的计算机的体系结构的变化.从图 1 中我们可以看到超算的体系结构从早期的向量处理器和单核通用 CPU 时代开始,然后演进到多核 CPU 时代,超级计算机的计算能力也从GFLOPS提升到TFLOPS.2008年的超算 Roadrunner[4]标志着异构架构的超级计算机体系结构的兴起,随后异构架构开始频繁涌现,超级计算机进入了 PFLOPS 的时代.各种加速器,如 GPU、DSP、FPGA、MIC 等都作为加速卡出现在各种超算中,我国有也了自己的国产加速器和国产处理器.伴随着超算体系结构的变化,HPL 算法的设计与实现也应该适应新的体系结构.在这种背景下,本文建立了国产处理器-国产加速器异构架构上的 HPL 性能模型,重点研究了国产处理器-国产加速器异构架构下的 HPL 算法的设计与实现.本文的主要贡献如下:

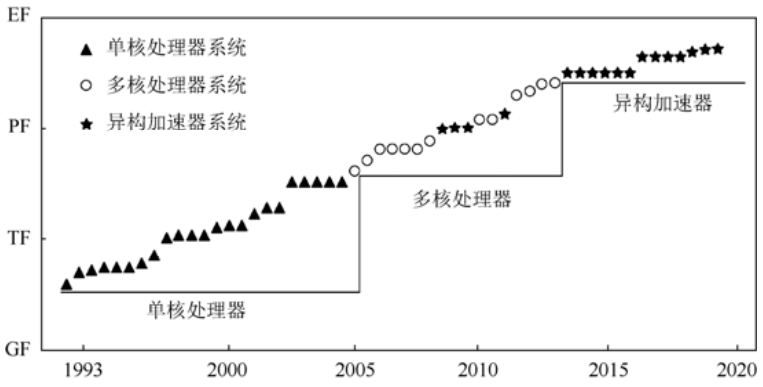

Fig.1 History of supercomputer architecture and computing power

图 1 超级计算机体系结构与计算能力发展历史

- 本文建立了一个 HPL 性能模型.
- 针对国产处理器-国产加速器异构架构,本文设计了一种多线程细粒度流水 HPL 算法,充分利用异构系统中的多种硬件,取得了超过同类系统其他实现的效率.
- 本文实现了轻量级跨平台的异构加速框架 HPCX.

本文第1节分析相关研究.第2节介绍HPL的性能模型.第3节给出针对异构架构的多线程细粒度流水HPL算法及异构加速框架 HPCX.第 4 节呈现新 HPL 算法的性能以及分析.第 5 节总结全文.

1 相关研究

HPL 是横量超算性能最重要的基准测试程序之一,其性能作为 TOP500 排名的依据.由于 HPL 的重要地位,国内外有许多针对 Linpack 的优化工作.这些工作主要集中在 HPL 算法优化,双精度矩阵乘法效率优化和 HPL 性能模型 3 个方面.

在 Linpack 算法优化方面,Dongarra 等人基于消息通信接口(message pass interface,简称 MPI)和基本线性数学库(basic linear arithmetic subroutine,简称 BLAS)的 Level 3 函数实现了分布式的 HPL 算法[1].在此基础上,文献[5]提出了一种单边通信和动态 look-ahead 算法,通过重叠第 i+1 轮迭代的 panel 分解与第 i 轮迭代的尾矩阵更新来实现部分通信与计算的重叠.针对异构架构,Fatica 通过将尾矩阵划分为固定比例的两块分别分配给加速器和 CPU 以实现 CPU 和加速器的协同计算[6].Yang 扩展了 Fatica 的工作实现了动态的 CPU 和加速器计算任务划分,采取根据上一次计算的 CPU/加速器性能比决定下一轮任务划分比例的策略[7].文献[8]实现了一种 work-stealing 的策略来实现 CPU 和加速器的计算任务动态平衡,并且利用有向无环图来维护算法中的计算依赖关系.通过这种对依赖关系的分析,文献[8]将行交换的过程分成几个部分,通过行交换和尾矩阵更新的相互重

© 中国科学院软件研究所 http://www.jos.org.cn

叠探索了一种粗粒度的流水线算法.

作为 HPL 的核心运算,双精度矩阵乘法的优化在 HPL 的优化中占有重要地位并且得到了全面而深入的研究.在传统 CPU 平台上,文献[9]给出了 CPU 上的矩阵乘法的分层算法,通过对 CPU 存储层次的模拟设计出相应的多级分块缓存策略,以尽可能地利用高速缓存中的数据.在包含加速器的异构架构上,李佳佳等提出了五阶段流水的异构矩阵乘算法来掩盖 CPU 与加速器之间的 PCIe 数据传输[10,11].MAGMA[12]通过细粒度的任务划分并且灵活地在CPU和加速器上调度这些任务来实现负载的动态平衡.为充分利用加速器如GPU的计算能力,文献[13]通过微基准测试来探测 GPU 的体系结构,在汇编语言层面做了多种优化,以实现接近 GPU 理论浮点峰值性能的双精度矩阵乘效率[13].

关于 HPL 的性能建模,文献[14]以预测 HPL 的扩展性为目的给出了 HPL 在 CPU 系统上的性能模型.HPL 的求解时间被建模为 panel 分解,panel 广播,行交换和尾矩阵更新的时间之和,并给出了模型中一些常量系数的经验值.王申等人在这个基础上考虑了look-ahead算法中部分广播开销可以被尾矩阵的更新和行交换所掩盖的情况[15],给出了更为精确的模型.文献[16]认为上述模型的计算都不够精确,因为模型中的常量值都是经验值,而 CPU 计算效率以及通信带宽等模型常量都会受数据量大小的影响.他们主张将这些常量系数视为可变的,通过已有的测试结果去学习这些系数,用学习得到得系数建模预测大规模求解的性能.

已有文献中对 HPL 的优化主要集中在同构架构上的简单算法优化和性能建模以及异构架构上的双精度矩阵乘的优化上,而缺少异构架构上的 HPL 的算法建模和算法流水层面的优化.本文在 CPU 的 HPL 性能模型的基础上建立了国产处理器-国产加速器异构架构上的HPL性能模型,并提出了多线程细粒度的HPL流水线算法,以充分发挥异构系统中国产处理器的巨大计算能力.在实现上,我们实现了一个轻量级的跨平台异构加速框架 HPCX,并用生产者消费者模型来协调多线程和多流的协同计算.

2 HPL 算法和性能模型

2.1 HPL算法简介

HPL 算法通过迭代法求解 N 阶线性方程组 $Ax=b$.求解过程包含两个步骤,首先通过带行交换的高斯消元法对系数矩阵进行 LU 分解得到 $[A\ b]=[[L,U]y]$,然后进行三角回带求解 x.其中 LU 分解的计算量为 $2N^3/3-N^2/2$,三角回带的计算量为 $2N^2$[1].给定系统的理论浮点峰值性能和 HPL 的求解时间 T,系统的实测浮点峰值性能 $R_{\max}$ 可以表示为 $R_{\max}=(2N^3/3-N^2/2)/T$,HPL 的效率 $E=R_{\max}/R_{\text{peak}}$.

$$A=\begin{pmatrix}A_{11} & A_{12}\\ A_{21} & A_{22}\end{pmatrix}=\begin{pmatrix}L_{11} & \\ L_{21} & L_{22}\end{pmatrix}\begin{pmatrix}U_{11} & U_{12}\\ & U_{22}\end{pmatrix}=\begin{pmatrix}L_{11}U_{11} & L_{11}U_{12}\\ L_{21}U_{11} & L_{21}U_{12}+L_{22}U_{22}\end{pmatrix} \tag{1}$$

HPL 的两个步骤中 LU 分解的计算量为 $O(N^3)$,三角回带的计算量为 $O(N^2)$,相比于 LU 分解的时间,三角回带的时间基本可以忽略[17],所以我们的优化也集中在 LU 分解上.式(1)给出了 LU 分解的符号表示,详细的算法数学证明请参考文献[1].在实现上,LU 分解以 NB 列为迭代步进行迭代求解,算法 1 给出了 LU 分解的详细算法描述.每一轮迭代包含 4 个子过程,分别是 panel 分解、panel 广播、行交换和尾矩阵更新.其中 panel 分解(panel_factorization)通过递归高斯消元求解得到 L_{11},U_{11} 和 L_{21};panel 广播(panel_bcast)将 L_{11} 和 L_{21} 广播给同行的行进程;行交换(row_swap)根据 panel 广播收到的行交换信息做行交换;尾矩阵更新(update)首先执行双精度三角矩阵求逆(DTRSM)得到 U_{12},然后通过将 L_{21} 和 U_{12} 做双精度矩阵乘(DGEMM)更新 A_{22} 矩阵.

算法 1. 并行 LU 分解算法.

1: for i=0; i<N; i+=NB **do**
2: panel_factorization(A,i,NB); // recursive panel factorization using Gaussian elimination
3: panel_bcast(L_{11},L_{21}); // broadcast panel in row processes, including row swap information
4: row_swap(A_{12},A_{22}); // pivoting according to swap information
5: update(A_{22},L_{21},U_{12}); // first calculate U_{12} by DTRSM, then update A_{22} using DGEMM
6: **end for**

© 中国科学院软件研究所 http://www.jos.org.cn

2.2 HPL性能模型.

在已有的 CPU HPL 建模分析的基础上[14,16–19],我们提出了适应于处理器-加速器异构架构的 HPL 性能模型.在具体介绍我们的 HPL 性能模型之前,我们先给出一些符号及其含义,大部分符号采取与文献[17]中一致的名称.矩阵 A 是 $N\times(N+1)$的系数矩阵,以 $NB\times NB$ 的块大小均匀分布在 $P\times Q$ 的二维进程网格中,$mp\times nq$ 表示每个进程处理的子矩阵的大小. f_{fact} 表示 panel 分解中浮点操作的比例,P_{cpu} 和 E_{cpu} 分别表示国产处理器双精度浮点峰值性能和浮点操作的效率,P_{acc} 表示国产加速器双精度浮点峰值性能,E_{dgemm} 和 E_{dtrsm} 分别表示国产加速器上 DGEMM 和 DTRSM 的效率,网络延迟为 Lat s,带宽为 BW byte/s.对于 panel 分解子过程,我们通过估计子过程中矩阵乘的计算量乘以一个系数来估计整体的浮点计算量,式(2)给出了这一子过程的时间估计.其中需要特别指出的是,我们将 panel 分解中大矩阵放到国产加速器上进行计算,这在式(2)的分母中体现了出来.

$$T_{\text{fact}}=\frac{f_{\text{fact}}\times mp\times NB\times NB}{P_{\text{cpu|acc}}\times E_{\text{cpu|dgemm}}} \tag{2}$$

对于 panel 广播算法的选择,我们实现中采用的是 HPL 软件包中复杂度较低的 Long 算法.它的复杂度为 log 级别,体现在式(3)前半部分的系数.每次广播,我们需要传输大小为 $NB\times NB$ 的 L_{11} 和 $mp\times NB$ 的 L_{12} 以及少量索引,这些数据均为双精度类型(8 字节),公式的后半部分给出了每一跳(从一个节点传往下一个节点)的时间估计.式(3)给出了对这一过程的时间估计.

$$T_{\text{bcast}}=\left(\frac{\log_2 Q}{2}+\frac{1}{Q}+1\right)\times(Lat+8\times\frac{mp\times NB+NB\times NB+NB+1}{BW}) \tag{3}$$

行交换时间的估计方式与 panel 广播的估计方式类似,区别在于采取的算法和传输的数据量不同.出于避免冗余数据传输的目的,行交换采用的是 spread-roll 算法[11],式(4)给出了这一子过程的时间估计.

$$T_{\text{swap}}=(\log_2 P+P-1)\times Lat+\left(\frac{\log_2 P}{2}+2\right)\times 8\times nq\times\frac{NB}{BW} \tag{4}$$

尾矩阵更新的过程主要是在国产加速器上执行两个 BLAS 库函数 DGEMM 和 DTRSM,其计算量分别是 $2\times mp\times nq\times NB$ 和 $nq\times NB\times NB$.式(5)给出了尾矩阵更新的时间估计.

$$T_{\text{update}}=\frac{2\times mp\times nq\times NB}{P_{\text{acc}}\times E_{\text{dgemm}}}+\frac{nq\times NB\times NB}{P_{\text{acc}}\times E_{\text{dtrsm}}} \tag{5}$$

性能模型中参数的值,我们分为两类.一类是可以预知的,比如问题的规模,分块的大小以及硬件的峰值浮点性能等等,对于这一类参数,我们根据系统硬件以及求解问题的实际规模设定好对应的值;另一类是不可以预知的,比如双精度矩阵乘的效率可能和矩阵的规模相关,网络的实际带宽和延迟可能受发送的数据量的影响等等,对于这一类参数,我们通过小规模实际测试给出其实测值.

我们用 TOP500 榜单中排名靠前的与国产处理器-国产加速器类似的异构系统,如 Summit[20]、Serria[21]、ABCI[22]以及曙光 E 级超算原型机对上述 HPL 性能预测模型进行了检验,结果见表 1[17].在大规模系统 HPL 性能预测的准确性上,最大误差值不到 5%.可以看到,我们建立的国产处理器-国产加速器异构 HPL 性能模型较为准确,可以给将来 E 级机的建造提供参考.

Table 1 TOP500 supercomputer performance prediction

表 1 TOP500 超级计算机性能预测

TOP 500	超算	节点配置	网络带宽 (Gb/s)	节点数	峰值性能 (PFLOPS)	实测性能 (PFLOPS)	预测性能 (PFLOPS)	误差
v	Summit	2*IBM POWER AC922 6*NVIDIA V100	200	4 608	200.8	143.5	140.6	<2%
2	Sierra	2*IBM POWER AC922 4*NVIDIA V100	200	4 320	125.7	94.6	92.8	<2%
7	ABCI	2*Intel Xeon Gold 6148 4*NVIDIA V100	200	1 088	32.6	19.8	20.8	<5%
	曙光 E 级超算原型机	2*国产处理器 1*国产加速器	200	512	3.2	2.3	2.2	<5%

© 中国科学院软件研究所 http://www.jos.org.cn

3 多线程细粒度 HPL 算法设计及实现

在已有的文献中,LU 分解的 4 个子过程,panel 分解,panel 广播,行交换和尾矩阵的更新都是顺序执行的.在过去纯 CPU 时代,由于尾矩阵更新占据了 90%以上的时间,HPL 的效率主要由 DGEMM 的效率决定,其他 3 个子过程对性能的影响不大.此时这种顺序执行 4 个步骤,或者通过简单的 look-ahead 算法[5]实现粗粒度流水的算法也能取得很好的效果.但是对于国产处理器-国产加速器异构架构,由于国产加速器计算能力与国产处理器的计算能力存在 1~2 个数量级的差距,尾矩阵更新的时间占比减少到了 50%左右,此时 panel 分解,panel 广播和行交换对性能的影响就不能忽略.在这样的背景下,探索一个新的细粒度流水算法用 update 的有用计算去掩盖 panel 分解、panel 广播和行交换的开销对于提升 HPL 的效率,充分发挥国产加速器的强大计算能力显得尤为重要.

3.1 多线程细粒度HPL算法的设计

HPL 耗时最多的计算是尾矩阵 A_{22} 更新的矩阵乘法计算,异构 HPL 算法加速的核心是利用国产加速器加速矩阵乘法.传统的 CPU-加速器异构 HPL 算法通过把 panel 分解的结果 L_{11},U_{12},L_{21} 矩阵拷贝到加速器内存,同时将更新前的尾矩阵 A_{22} 拷贝到加速器内存,利用加速器求解 U_{12} 和更新尾矩阵,将更新后的尾矩阵 $\widetilde{A_{22}}$ 拷贝回 CPU 内存[17].这种做法将系数矩阵放在 CPU 端内存中,每次调用加速器的 DGEMM 都需要把数据通过 PCIe 拷贝到加速器内存,在完成计算后又需要把结果矩阵拷贝回 CPU 内存.在文献[6]中通过三阶段流水的办法用加速器上的计算来掩盖 PCIe 数据传输的开销,但是加速器算力增加的速度远远高于 PCIe 带宽的增加速度,它们之间越来越大的差距使得加速器计算的时间无法掩盖 PCIe 传输的时间.为了解决这个问题,我们将系数矩阵放在国产加速器的内存上,这样就避免了国产处理器和国产加速器之间大量的数据交换.只需要在国产处理器做 panel 分解之前,从国产加速器把 panel 需要的 NB 列数据拷贝回来就可以了.假设当前迭代中剩余待求解系数矩阵大小为 $n\times(n+1)$,原来粗粒度并行的算法中,我们需要通过 PCIe 移动 $8\times(2\times n\times(n+1)+n\times NB+NB\times(N+1))$ 字节的数据,现在只需要移动 $8\times(2\times n\times NB+NB\times(n+1))$ 字节数据,通过 PCIe 的数据传输量大大减少了.这个版本的 HPL 算法我们称为粗粒度 HPL 算法.

粗粒度 HPL 算法存在两个问题导致其不能取得很高的性能.一个问题是由于尾矩阵更新时间占比减少,行交换的网络传输的开销显得越来越大.另一个问题是通过简单使用国产加速器的异步流机制让国产处理器端的 panel 分解和国产加速器端的 update 并行,国产处理器与国产加速器只有很弱的并行工作的效果,大部分时间国产处理器与国产加速器都是串行执行,这造成了国产加速器大量的空闲等待时间.为了解决这两个问题,我们设计了一种国产处理器-国产加速器异构多线程细粒度流水算法.我们通过对数据依赖的分析发现尾矩阵更新与行交换在列与列之间是没有数据依赖的.受此启发,我们在列方向上对尾矩阵进行分块,如图 2 所示,将完整的尾矩阵行交换和更新划分成一个个由若干 NB 列块组成的单元进行行交换和更新.行交换主要利用 PCIe 和网络,对国产加速器的计算资源占用率不高,这样就用尾矩阵更新的计算掩盖了行交换的开销[17].在上面细粒度任务划分的基础上,我们引入了多线程多流机制来协调国产处理器与国产加速器的计算.具体来说,我们引入了 4 个线程,如图 3 所示,thread 0 负责 panel 分解和 panel 广播,thread 1 负责 PCIe 的数据传输,thread 2 负责行交换,thread 3 负责尾矩阵更新.thread 1 和 thread 2 两个线程运行在同一个国产处理器物理核心上,thread 0 和 thread 3 分别运

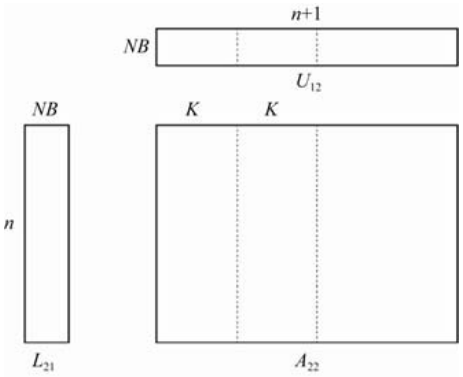

Fig.2 HPL fine-grained parallel data splitting (K is a multiple of NB)

图 2 HPL 细粒度并行数据划分(K 是 NB 的倍数)

© 中国科学院软件研究所 http://www.jos.org.cn

行在其他两个国产处理器物理核心上.除线程 0 外,每个线程管理各自的异步流.通过利用线程间同步和流之间的同步来协调国产处理器与国产加速器的计算,最终实现了如图 3 所示的流水线.

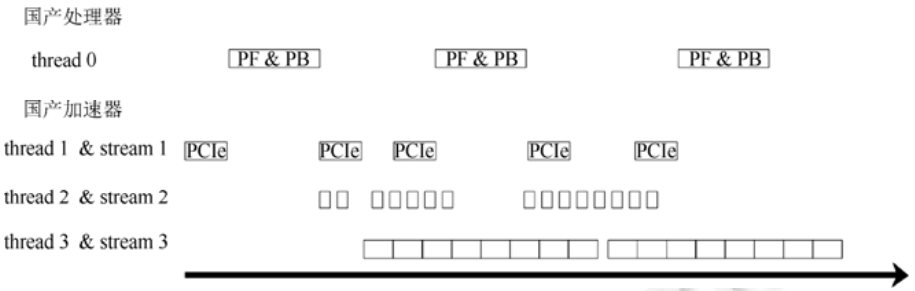

Fig.3 HPL multithread fine-grained parallel algorithm flow

图 3 HPL 多线程细粒度并行算法流程图

3.2 多线程细粒度HPL算法的实现

我们通过引入简单的生产者消费者模式来维护细粒度算法的依赖关系,以降低多线程带来的开销,实现与文献[17]中做法一致.如图 4 所示,行交换生产者做完一个列块的行交换之后,生成一个更新任务放到更新任务队列里边;负责尾矩阵更新的线程作为消费者,取出更新队列里面的任务并执行,同时尾矩阵更新线程还是传输任务的生产者,在执行完一个更新任务后,生成一个传输任务放到传输任务队列里边;负责传输的线程从传输队列里取任务完成传输;负责 panel 分解的进程在等待自己需要的列块数据更新完成之后就可以并行开始做下一轮的 panel 分解[17].各个线程间利用信号量实现等待和唤醒,当任务队列为空的时候,相应线程就挂起,避免忙等待带来的开销.

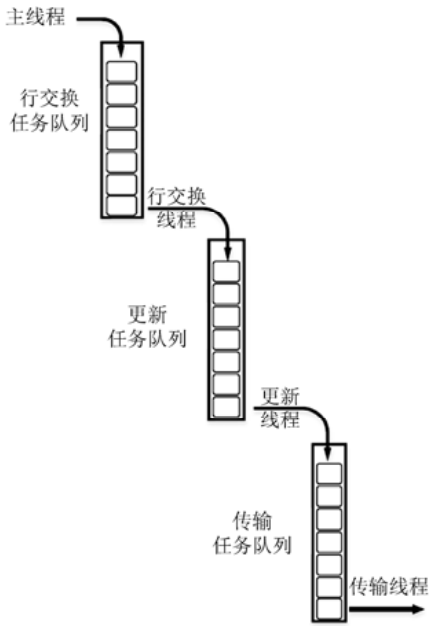

Fig.4 HPL producer-customer model

图 4 HPL 生产者消费者模型

异构系统的加速器有多种,比如 GPU、MIC、FPGA、国产加速器等等.为了让异构 HPL 算法具有可移植性,能够运行在多种异构平台上,我们完成了一个轻量级的异构加速框架 HPCX[17].其实现与文献[17]一致,如图 5 所示,我们抽象出了异构加速平台的一些共有特性,比如内存管理,并行计算,数据传输,异步调用等等.同时我们对不同厂商的异构加速器编程模型和基础数学库进行总结,定义了一套统一的编程结构.在不同的加速器上,使用不同的编程模型(HIP、CUDA、C)实现,底层用不同的编译器编译成不同平台上的二进制程序.目前 HPCX 支持国产加速器、AMD GPU 和 NVIDIA GPU 以及国产处理器、Intel CPU 和 AMD CPU.对于其他异构加速器,结合硬件平台给出 HPCX 定义抽象接口的具体实现就可以方便整理到 HPCX 框架中.异构并行 HPL 算法通过调用 HPCX 提供的编程接口实现跨平台加速.

© 中国科学院软件研究所 http://www.jos.org.cn

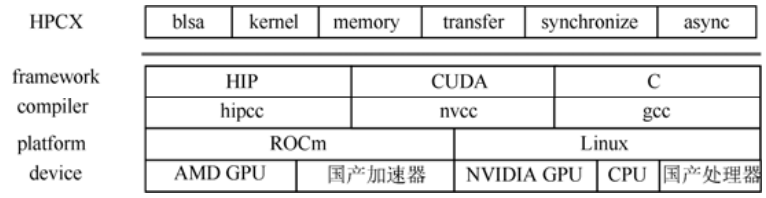

Fig.5 Heterogeneous acceleration framework HPCX

图 5 异构加速框架 HPCX

4 性能测试与分析

我们实现了包括国产加速器和 NVIDIA GPU 两种异构平台上的粗粒度版本 HPL 和多线程细粒度版本 HPL,在国产加速器和 NVIDIA 两个平台上进行了测试.在 NVIDIA 平台上,我们将我们实现的两个版本的 HPL 与开源成果三阶段流水线版本 HPL 以及目前 NVIDIA 平台上效率最高的 NVIDIA 官方非开源程序 nvhpl 进行了对比.在国产加速器平台上,我们在曙光 E 级超算原型机的 512 个节点上进行大规模扩展性测试.

4.1 实验平台简介

表 2 给出了我们实验平台的信息.在两种平台上,单个节点内处理器与加速器都通过 PCIe 3.0 总线连接.NVIDIA 平台上只有一个节点,配有两张 P100 显卡.国产加速器平台上有多个节点,每个节点上装有一个国产加速器,节点之间采用 100Gb/s 的 EDR 网络连接.

Table 2 Configuration of computing nodes

表 2 计算节点的配置

	国产加速器	NVIDIA
处理器	国产处理器	Xeon E5-2690
封装数	1	2
核心数	32	28
双精度浮点峰值(GFLOPS)	384	291.2
内存(GB)	64	128
加速器	国产加速器	P100
加速器个数	1	2
双精度浮点峰值(TFLOPS)	5.9	4.9
内存(GB)	32	16

4.2 性能与分析

在 NVIDIA 平台上的单卡测试结果如图 6 所示.从图 6 中我们可以看到,随着问题规模的变大,除了开源的 3 阶段流水线版本 HPL 的性能没有太大提高外,nvhpl 与我们实现的两个版本 HPL 均有明显性能提升.出现这种情况是因为 3 阶段流水线版本的 HPL 的矩阵位于 CPU 上,而三阶段流水线无法用加速器的计算掩盖 PCIe 数据传输的开销.我们可以看到,通过简单地把矩阵放到加速器内存上,粗粒度 HPL 就获得了很大的性能提升,说明将矩阵置于加速器内存上是合理的.但是粗粒度 HPL 算法与 nvhpl 相比还是有较大差距,原因是粗粒度 HPL 算法对加速器与处理器的并行度挖掘不够,以及忽略了行交换中网络通信开销的优化.多线程细粒度 HPL 算法在做完上述优化之后,性能完全超越了 nvhpl 的性能,平均领先 nvhpl 达 9%.图 7 展示了 NVIDIA 平台上多卡的测试结果.由于三阶段流水线版本 HPL 速度太慢,我们略去了它的多卡测试.从图 7 中我们可以发现,粗粒度 HPL、多线程细粒度 HPL 与 nvhpl 都有较好的扩展性.在多卡测试上,我们的细粒度版本 HPL 依然领先 nvhpl.

如图 8 所示,在曙光 E 级原型机的 512 个节点上,我们进行了 1~512 个节点的扩展性测试.从图 8 中可以看出,在不同测试进程规模下,HPL 的扩展性很好,随着节点的增加 HPL 的测试效率缓慢下降,从 2 个节点约 75% 的效率下降到 512 个节点约 71%的效率.需要注意图中单节点效率偏低是因为单节点测试采用的 NN 格式(非转置非转置)的矩阵乘,而多节点采用的 NT 格式(非转置转置)的矩阵乘,前者的效率低于后者.我们实现的多线

© 中国科学院软件研究所 http://www.jos.org.cn

程细粒度版本 HPL 最终在 512 个节点上实现了 HPL 实测峰值性能 2.3 PFLOPS,实测效率 71.1%优秀测试结果.

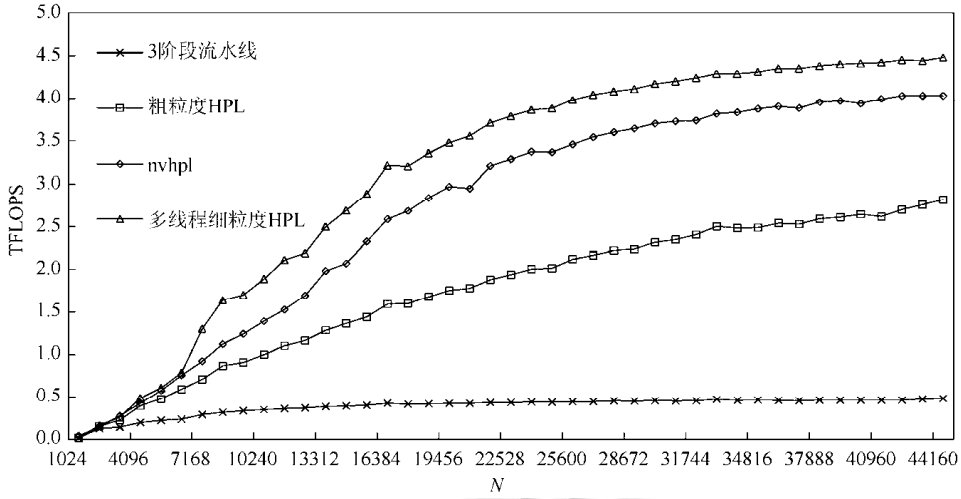

Fig.6 HPL performance on single NVIDIA GPU

图 6 NVIDIA GPU 单卡 HPL 性能

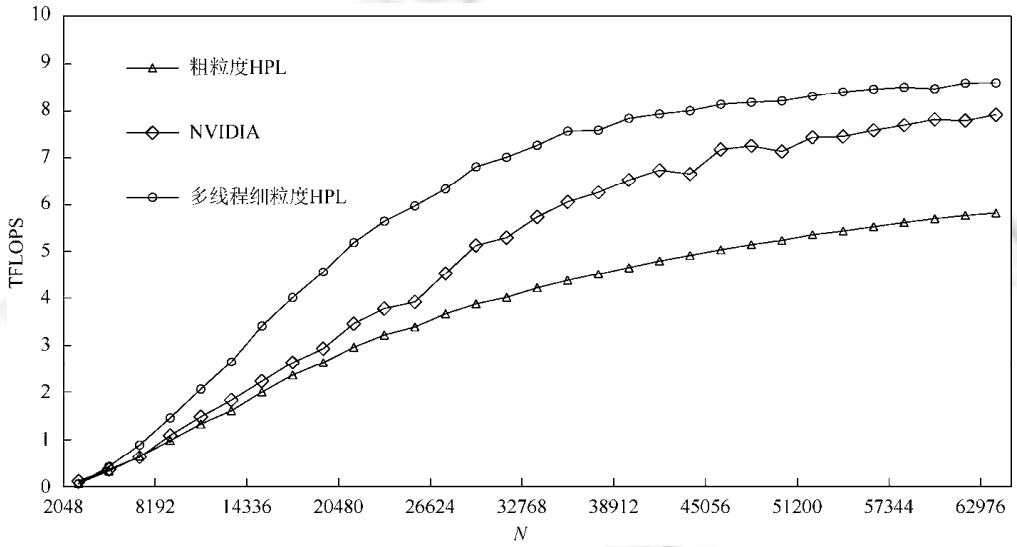

Fig.7 HPL performance on multiple NVIDIA GPUs

图 7 NVIDIA GPU 多卡 HPL 性能

© 中国科学院软件研究所 http://www.jos.org.cn

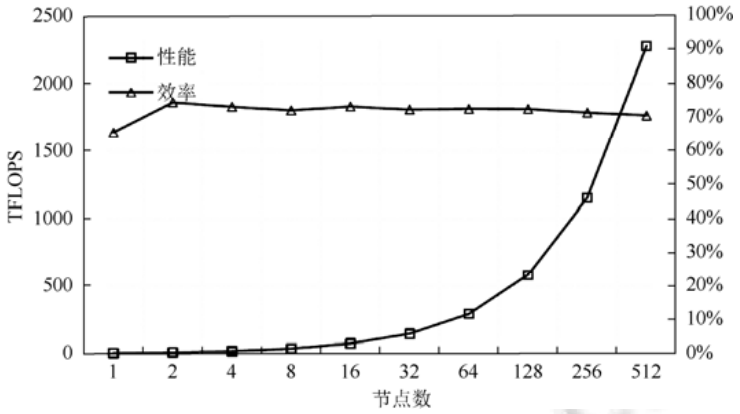

Fig.8 Sugon E-prototype supercomputer HPL performance

图 8 曙光 E 级超算原型机 HPL 性能

5 结 论

本文提出的异构 HPL 算法通过将矩阵存储于国产加速器的内存解决了数据传输瓶颈,通过多线程细粒度的算法软流水实现了对通信开销的掩盖,通过一个轻量级异构加速框架 HPCX 提供的对国产加速器的基本操作的抽象实现了跨平台的异构 HPL 算法.在同类异构系统上,我们实现的算法性能远远超过开源的工作,并且优于 NVIDIA 公司的非开源 HPL 程序.我们的算法也展示了良好的扩展性,在曙光 E 级超算原型机 512 个节点 HPL 测试中实现了 71.1%的效率.同时,我们的性能模型也展示了较高的准确性,可以为未来 E 级异构超算的 HPL 性能预测提供参考.

References:

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

[2] Walker DDW. Software libraries for linear algebra computations on high performance computers. SIAM Review, 1995,37(2): 151−180.

[3] TOP500. 2019. https://www.top500.org/

[4] Barker KJ, Davis K, Hoisie A, Kerbyson DJ, Lang M, Sancho SJC. Entering the petaflop era: The architecture and performance of roadrunner. In: Proc. of the 2008 ACM/IEEE Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC). IEEE, 2008. 1−11.

[5] Husbands P, Yelick K. Multi-threading and one-sided communication in parallel LU factorization. In: Proc. of the 2007 ACM/IEEE Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC). IEEE, 2007. 1−10.

[6] Fatica M. Accelerating linpack with CUDA on heterogenous clusters. In: Proc. of the 2nd ACM Workshop on General Purpose Processing on Graphics Processing Units (GPGPU-2). ACM, 2009.46−51.

[7] Yang CQ, Wang F, Du YF, *et al*. Adaptive optimization for petascale heterogeneous CPU/GPU computing. In: Proc. of the 2010 IEEE Int'l Conf. on CLUSTER Computing. IEEE, 2010. 19−28.

[8] Heinecke A, Vaidyanathan K, Smelyanskiy M, *et al*. Design and implementation of the linpack benchmark for single and multi-node systems based on Intel® Xeon Phi coprocessor. In: Proc. of the 2013 IEEE Int'l Symp on Parallel & Distributed Processing (IPDPS). IEEE, 2013. 126−137.

[9] Goto K, Geijn RAVD. Anatomy of high-performance matrix multiplication. ACM Trans. on Mathematical Software, 2008,34(3): 1−25.

[10] Li JJ, Li XJ, Tan GM, *et al*. An optimized large-scale hybrid DGEMM design for CPUs and ATI GPUs. In: Proc. of the 26th ACM Int'l Conf. on Supercomputing (ICS). ACM, 2012. 377−386.

© 中国科学院软件研究所 http://www.jos.org.cn

[11] Mittal S, Vetter JS. A Survey of CPU-GPU heterogeneous computing techniques. ACM Computing Surveys, 2015,47(4):Article 69.

[12] Tomov S, Dongarra J, Baboulin M. Towards dense linear algebra for hybrid GPU accelerated manycore systems. Parallel Computing, 2010,36(5-6):232–240.

[13] Tan GM, Li LC, Triechle S, Phillips E, Bao YG, Sun NH. Fast implementation of DGEMM on Fermi GPU. In: Proc. of the 2011 ACM/IEEE Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC). IEEE, 2011.1–11.

[14] Dongarra J, van de Geijn R, Walker D. Scalability issues affecting the design of a dense linear algebra library. Journal of Parallel and Distributed Computing, 1994,22(3):523–537.

[15] Li JJ, Li XJ, Tang GM. Research of DGEMM performance on CPU/ATI GPU heterogeneous architecture. Information Technology Letter, 2011,9(6):12–27 (in Chinese).

[16] Chou CY, Chang HY, Wang ST, Huang KC, Shen CY. An Improved Model for Predicting HPL Performance. In: Proc. of the 2nd Int'l Conf. on Advances in grid and pervasive computing. Springer-Verlag, 2007. 158–168.

[17] Yu XZ. An optimized large-scale hybrid DGEMM design for CPUs and ATI GPUs [MS. Thesis]. Beijing: Institute of Computing Technology, Chinese Academy of Sciences, 2019 (in Chinese with English abstract).

[18] Wang S, Qi FB, Gu HF, Pan Z. Linpack parallel performance model and its prediction. Computer Engineering, 2012,38(16):81–84 (in Chinese with English abstract).

[19] Zhang WL, Chen MY, Fan JP. Emulation and Forecast of HPL Test Performance. Journal of Computer Research and Development, 2006,43(3):557–562 (in Chinese with English abstract).

[20] Summit. 2019. https://www.olcf.ornl.gov/summit/

[21] Sierra. 2019. https://hpc.llnl.gov/hardware/platforms/sierra

[22] ABCI. 2019. http://abci.ai/

附中文参考文献:

[15] 李佳佳,李兴建,谭光明.CPU/ATI GPU 混合体系结构上 DGEMM 的性能研究.信息技术快报,2011,9(6):12–27.

[17] 于献智.面向 E 级计算的大规模 HPL 算法设计与实现[硕士学位论文].北京:中国科学院计算技术研究所,2019.

[18] 王申,漆锋滨,谷洪峰,潘治.Linpack 并行性能模型及其预测.计算机工程,2012,38(16):81–84.

[19] 张文力,陈明宇,樊建平.HPL 测试性能仿真与预测.计算机研究与发展,2006,43(3):557–562.

水超洋(1994－),男,博士生,主要研究领域为稠密矩阵乘法优化,稀疏张量优化.

王银山(1988－),男,博士,副研究员,CCF 专业会员,主要研究领域为数值模拟,大规模并行计算,稀疏矩阵计算优化.

于献智(1994－),男,硕士,主要研究领域为异构高性能计算.

谭光明(1980－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为并行算法设计与分析,并行编程和优化,计算机体系结构,生物信息学,大数据.

© 中国科学院软件研究所 http://www.jos.org.cn