软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(8):2329−2340 [doi: 10.13328/j.cnki.jos.006005]

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

面向异构计算机平台的 HPL 方案*

孙 乔[1], 孙家昶[1], 马文静[1,2], 赵玉文[1,3]

[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)

[3](中国科学院大学,北京 100049)

通讯作者: 赵玉文, E-mail: zhaoyuwen@iscas.ac.cn

摘 要: HPL(high performance Linpack)是一套被广泛用于测评计算机性能的测试程序,几十年来学术界及产业界十分关注对 HPL 测试程序的定制化优化工作,以充分反应同时代新兴计算机平台的性能.面向当今主流多设备异构计算平台,尝试为 HPL 的优化工作提供一种解决方案:Hetero-HPL.在 Hetero-HPL 中,进程与协处理器的对应关系可被改变,因此 HPL 算法在单节点独立运行情况下可以完全避免进程间数据传输开销.算法各个重要步骤有能力完全利用物理节点的所有资源,如内存容量、CPU 核心、协处理器、PCI-e 总线等.Hetero-HPL 并不引入冗余计算量及通信量,并在任意设备数量下妥善应对锁页内存分配限制,确保多设备负载均衡和设备内高效的大规模同质运算.在实验平台上,Hetero-HPL 效率可以达到平台峰值性能的 76.5%(其中,dgemm 函数效率为 84%).进一步的实验结果表明,Hetero-HPL 在多节点联机运行情况下也是一种可行的方案.

关键词: HPL(high performance Linpack);多设备异构平台;并行计算

中图法分类号: TP303

中文引用格式: 孙乔,孙家昶,马文静,赵玉文.面向异构计算机平台的 HPL 方案.软件学报,2021,32(8):2329−2340. http://www.jos.org.cn/1000-9825/6005.htm

英文引用格式: Sun Q, Sun JC, Ma WJ, Zhao YW. HPL approach for heterogeneous computer platforms. Ruan Jian Xue Bao/ Journal of Software, 2021,32(8):2329−2340 (in Chinese). http://www.jos.org.cn/1000-9825/6005.htm

HPL Approach for Heterogeneous Computer Platforms

SUN Qiao[1], SUN Jia-Chang[1], MA Wen-Jing[1,2], ZHAO Yu-Wen[1,3]

[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)

[3](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: HPL (high performance Linpack) is a widely used benchmark for measuring computer performance. Over the decades, the practice of optimizing and tuning of HPL has constantly drawn great attention in both industrial and academic circle, to evaluate the performance of contemporary cutting-edge computer platforms. For current heterogeneous HPC platforms with multiple accelerating co-processors, an approach of high-performance HPL benchmark, Hetero-HPL, is proposed in this paper. In Hetero-HPL, the mapping between process set and (co-) processor set becomes adjustable, so that the computation within each computing node may avoid inter-process message exchange, and each important procedure of the HPL algorithm may make full use of the hardware resources of the computing node, such as memory, CPU cores, co-processors, and PCI-e bus etc.Without redundant computation and communication, the

* 基金项目: 国家重点研发计划(2018YFB0204404); 中国科学院战略性先导科技专项(C 类)(XDC01030200)

Foundation item: National Key Research and Development Program of China (2018YFB0204404); Strategic Priority Research Program of the Chinese Academy of Sciences (Category C) (XDC01030200)

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

working set of Hetero-HPL is not restricted by the limit of pinned memory size in a single allocation, and is distributed in a way that the workload is balanced among all the co-processors and massive fine-grained parallelism can be exploited. On one experimental platform with four co-processors, Heter-HPL can reach an efficiency of 76.5% (the efficiency of function dgemm is 84%) in one computing node, and further experiment suggests that Hetero-HPL is also a feasible approach in distributed environment.

Key words: HPL (high performance Linpack); multi-device heterogeneous platform; parallel computing

1 HPL 基准测试及主流异构并行架构

HPL(high performance Linpack)[1]是一套被广泛用于测量计算机实际峰值计算性能的基准测试程序(benchmark).以其实际运行性能为标准,国际超级计算机性能排行榜 TOP-500[2]每年会对众多超算进行性能排名.例如,2019 年,来自美国的超算“Summit”[3]以实测 HPL 148.6 PFLOPS 双精度浮点性能位居 TOP-500 榜首,而来自中国的神威-太湖之光则以 93.0 PFLOPS 位居第三.TOP-500 排行榜是衡量全世界超级计算机发展的重要指标,而其核心测试程序 HPL 的性能优化问题历来是各个超算厂商乃至各个国家发展高性能计算事业所关注的重点.在不断提高 HPL 运行效率的过程中,程序的性能表现也为计算机系统及其基础软件的开发者提供有效的反馈信息,能够有效促进计算机系统的向前发展.

由于众核协处理器(如 GPGPU)的普及,目前高性能计算机一般采用多路(socket)多核 CPU+众核协处理器架构,如在 2019 年 TOP-500 榜单中名列前茅的 Summit、Sierra 及天河 2 号等.CPU 和协处理器往往采用不同的体系结构及指令集,这样的计算机架构被称为异构计算机架构.在这样的系统中,传统多核 CPU 负责执行程序的逻辑密集部分,众核协处理器则高效地处理程序中计算密集的部分.由于协处理器的高性能主要依赖于大量轻量级核心提供的并行处理能力,因此可大幅度提高计算机系统整体的计算功耗比.鉴于异构众核架构诸多优势,其如今已在事实上成为了当今高性能计算机建设的主流解决方案.

图 1 展示了目前典型的异构众核架构,从总体上来说,该架构可以简单地区分为 Host 端和 Device 端.Host 端采用的 CPU 具有多个物理上分开的 Socket,而每个 Socket 中有若干物理核心.这些核心是常规 CPU 核心,能够有效处理程序中复杂的逻辑判断部分,因此程序的主进程运行在这些核心上.从内存层次结构上看,当今多路多核 CPU 一般具有多级数据和指令缓存.如图 1 所示,每个 CPU 核心具有独立的一级缓存(*L*1-cache),而处于同一 Socket 的若干 CPU 核心共享二级 Cache(*L*2-cache),可有效加速多个 CPU 核心间共享数据或指令的访问.而作为 CPU 与内存之间的桥梁,三级 Cache(*L*3-cache)用来提高 CPU 整体的访存速度.从图 1 中还可以看到,异构众核系统的内存和 CPU 的各个 Socket 一样也是物理上独立的,每个 Socket 都有与自己邻近的一块物理内存,该 Socket 上的 CPU 核心能够以较低的延迟访问近端内存,但若访问远端内存则延迟大幅度提高.各个 Socket 直接由高速网络互联,并在硬件上实现了访存一致性协议,形成 cc-NUMA(cache coherent non-uniform memory access)结构,使得各个 CPU 核心能在逻辑上共享一致的内存空间.值得注意的是,缓存一致性协议及远端内存访问会带来较大的开销,一种比较通用的方法是将各个进程分布在不同的 Socket 上使得各个进程仅利用与该 Socket 毗邻的资源.

目前 Device 端可以配备多个协处理器设备.这些协处理器相对于 CPU 来说具有很高的计算能力.这样强大的计算能力一般是通过大规模并行计算实现.以当今主流的 GPGPU 为例,一块 GPGPU 卡中包含了众多流处理器,这些处理器结构相对简单而不适合处理逻辑复杂的代码,但这些流处理器在硬件上整合了大量的向量计算部件并实现了高效的线程切换机制.这样,一个流处理器能够高效地调度大量线程予以向量化并行执行.协处理器拥有独立的内存空间,一般通过 PCI-e 总线与 Host 端的内存进行数据交换.

虽然上述异构架构较为普及,但以往对 大规模超级计算机上 HPL 测试程序的优化工作的开展大多基于进程与协处理器一一对应这一前提[4−6].这种方案易于优化,也可避免缓存一致性协议及访问远端内存带来的开销.但在 HPL 程序的实际执行过程中,这种方案将不可避免地导致大量处理器空闲,并增加结点内进程间通信开销.在本文中,我们面向高度异构架构提出一套新型 HPL 测试程序 Hetero-HPL,突破以往进程与协处理必须一一对应这一限制,使用进程内多协处理级并行方案,进而研究基于此方案的性能优化方法,在执行过程中更充分地

利用各种硬件资源.

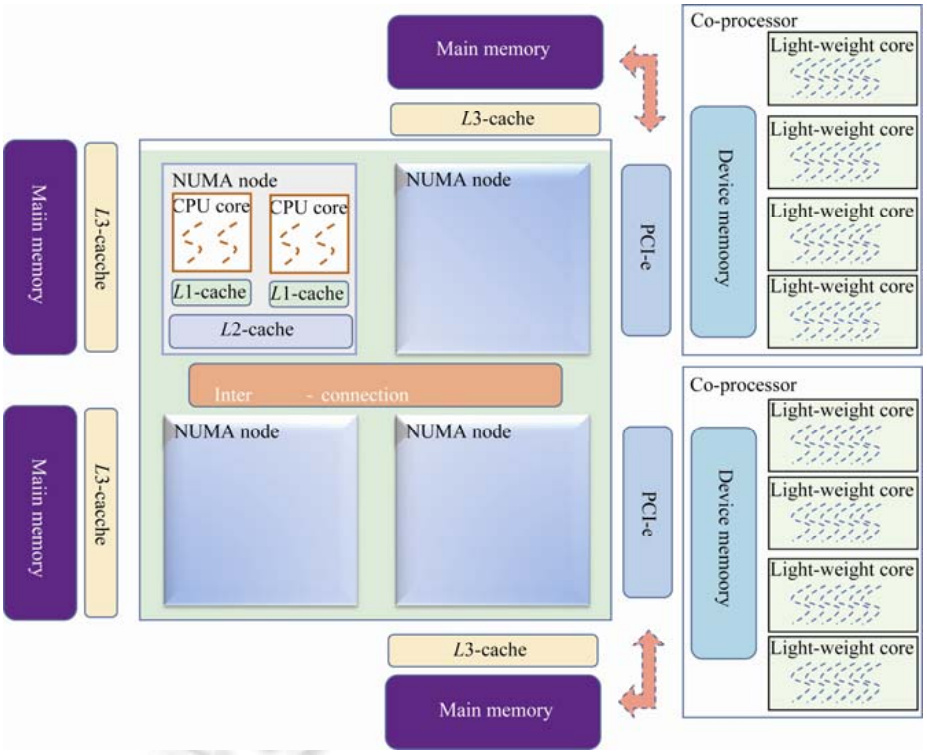

Fig.1 Typical architecture of multi-device heterogeneous platform

图 1 典型多设备异构平台架构

2 HPL 算法概要

本节仅简要介绍 HPL 程序执行过程中与本文工作相关的大致流程,关于各种算法细节可以参考[3,7−9]等,在此不再赘述.HPL 测试程序以多进程分布式的方式求解线性方程组 $Ax=b$.矩阵 A 以二维循环块卷帘(block cyclic)的方式分布在二维矩形进程拓扑上.由于进程拓扑设置具有自由性,而 HPL 测试常常基于正方形进程拓扑结构而展开,因此本文仅讨论典型的正方形进程拓扑下的 HPL 分解算法,但本文工作也可用于非正方形进程拓扑结构的 HPL 测试.

HPL 程序首先采用部分选主元(patial pivoting)的方式对随机矩阵 A 进行 LU 分解,之后进行回代求解向量 $\boldsymbol{b}$.由于其绝大部分计算量集中在对矩阵 A 的 LU 分解部分,对 HPL 算法的性能优化工作主要围绕该部分展开.对矩阵 A 的 LU 分解操作从逻辑上可以分为 Panel 分解及 Update(尾子矩阵更新)两个部分.图 2 展示了 HPL 程序在 3×3 进程拓扑下的大体执行流程.

在图 2 中我们可以看到,对矩阵 A 的 LU 分解过程以由多个迭代步骤完成.在每次迭代过程中,首先处于某一特定列(简称 P-Fact 列)的所有进程协作完成 Panel 的分解.Panel 分解完成对当前瘦高子矩阵(panel)的部分选主元过程的 LU 分解操作.从具体算法上看,Panel 分解可选择向左或向右看(left/right looking)或 Crout LU 分解算法.这三者计算量大致相等,具体的选择可根据实际运行的效率决定.P-Fact 列的进程在 Panel 分解执行之后,通过数据传输,将产生的行交换信息(dpiv),Panel 分解的结果矩阵($L1$ 和 $L2$)分享给同一进程行中的其他进程列,并开始接下来的 Update 操作.在此值得一提的是,在 Panel 分解过程所涉及的计算从形式上说具有逻辑复杂,小规模反复执行等特点,并行度较小.因此线程启停开销,函数调用,数据传输开销将严重制约 Panel 分解过程的效

率.因此其不适合采用 Device 端加速设备执行.

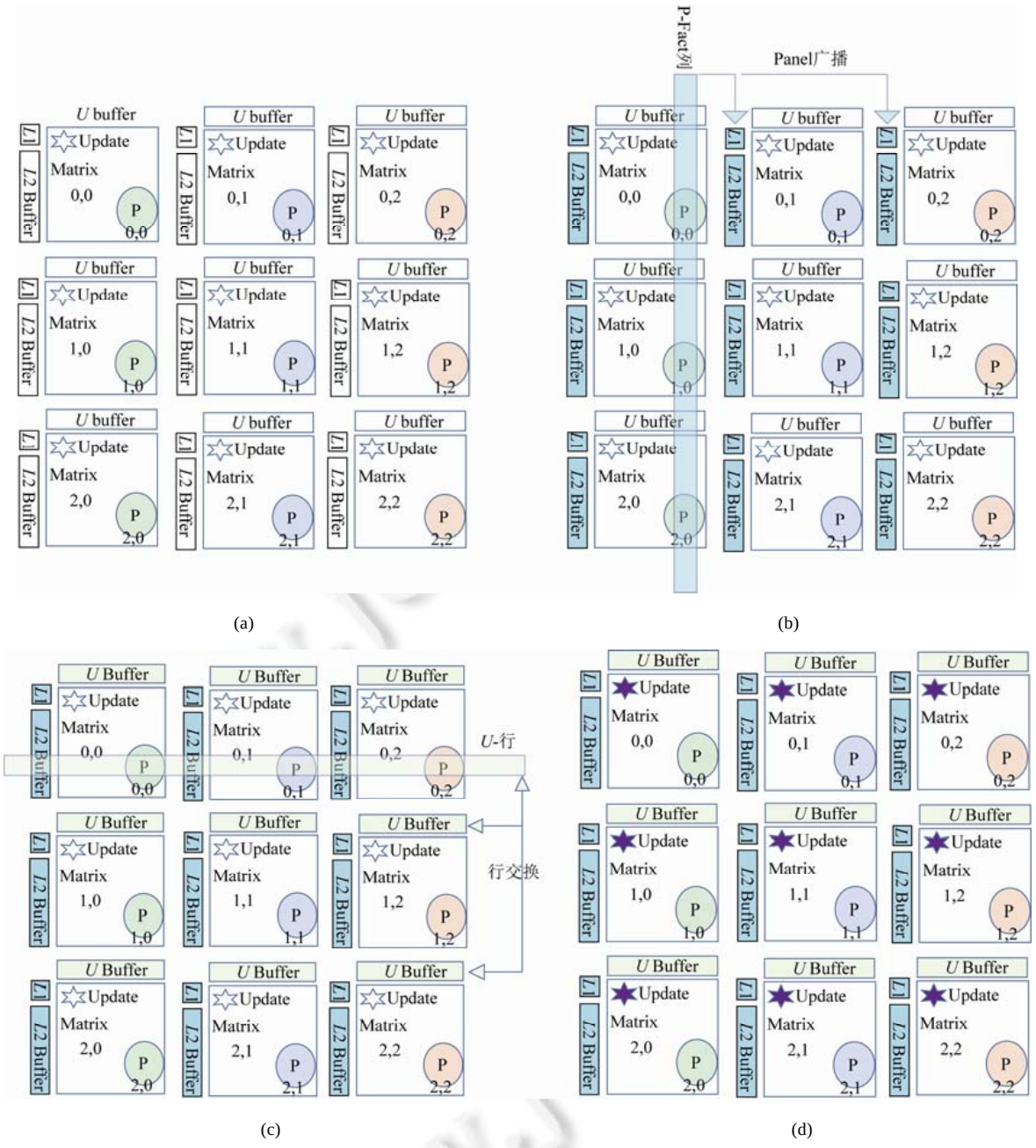

Fig.2 One iteration of HPL algorithm

图 2 HPL 算法单次迭代

在 Panel 分解执行完毕之后,所有列的进程开始执行 Update 过程.在该过程中拥有矩阵 *U*(或其某一部分)的进程行我们称为 *U* 行进程.这些进程将 *U* 中待交换的矩阵行集合后配合其他行进程完成矩阵对应列的分布式行交换过程,其结果是每一进程都拥有自身矩阵对应所需的 *U* 矩阵.之后,参与 Update 操作的所有进程分别利用得到的 *L*1 矩阵对交换后的 *U* 矩阵进行 dtrsm(三角矩阵求解)更新,并利用得到的 *L*2 矩阵和矩阵 *U* 对所属自

身的尾子矩阵调用 dgemm(矩阵乘法)完成更新.

不论进程与 CPU 以怎样的方式对应,HPL 算法中占据绝大部分计算量的尾子矩阵更新操作都能完全利用所有协处理器完成.在该过程通过协处理进行充分加速之后,其余的零散计算的资源使用率将成为性能瓶颈.其中典型的有 Panel 分解操作及处于 Update 阶段的分布式行交换操作.从算法流程上,在 HPL 的每一次迭代步骤中,只有特定 P-Fact 列进程完成本次迭代所需的 Panel 分解操作,同时也只有 *U* 行进程参与矩阵 *U* 的广播.因此不同的进程与处理器的映射关系会导致不同的资源使用情况.图 3 展示了一个由 4 个实验节点(具体参数介绍见第 5.1 节)组成的分布式环境.图 3 左侧展示了进程与 Socket 一一对应的情况.由于每一个 Socket 仅管理 8 个 CPU 核心和一个协处理设备.因此在 P-Fact 过程中只有共计 8×4=32 个 CPU 核心参与 Panel 分解运算,而在 Update 阶段的行交换过程中,仅有 4 个协处理器完成对 *U* 矩阵的打包并由对应的 4 道 PCI-e 总线进行传输.若采用图 3 右侧进程与节点(node)一一对应的方式执行 HPL 程序,每个进程管控 4 个协处理器设备,则在 Panel 分解部分将有 8×4×2=64 个 CPU 核心参与计算,而在 Update 阶段的行交换过程中会有 8 个协处理器及其 PCI-e 总线参与.随着协处理器运算能力的进一步增长,占绝大部分计算量的 BLAS 3 级函数能够很快完成,而诸如 Panel 分解,数据打包等必要步骤的运行时间比重将进一步加大,因此应在这些步骤中应尽可能地充分利用各种硬件资源.

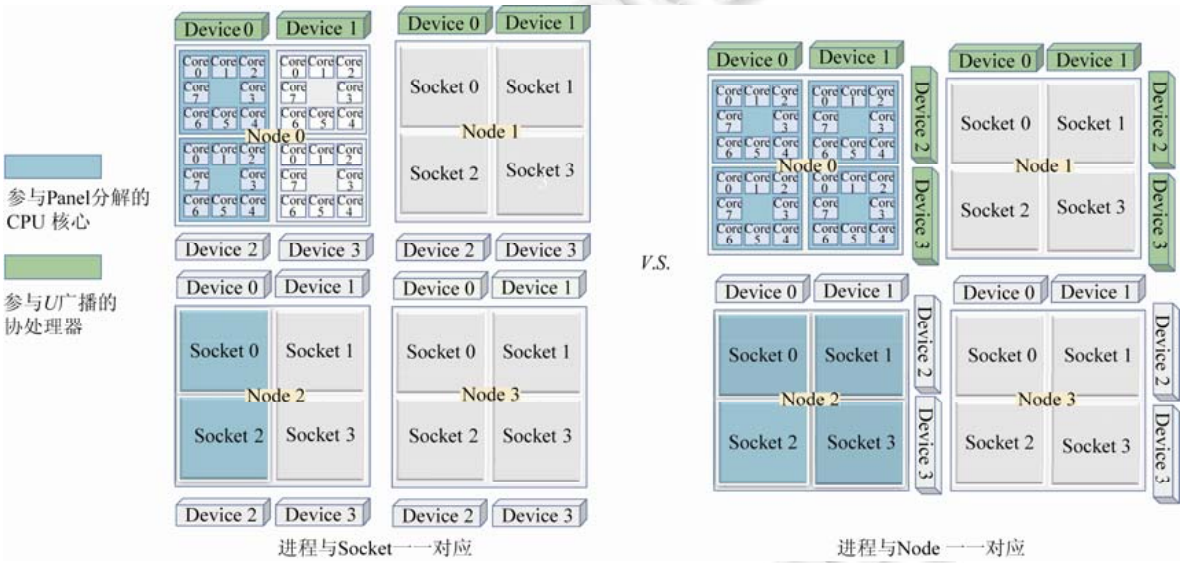

Fig.3 Different mappings between processes and nodes

图 3 进程与节点的不同对应关系

3 Hetero-HPL 总体结构

为了提高 HPL 各个阶段的资源利用率.本文提出 Hetero-HPL 方案.总体上,Hetero-HPL 中每一个进程能够管理任意数量的协处理器,在大幅度提高上述关键计算步骤的资源利用率的前提下,以协处理器间高度负载均衡的方式完成 HPL 计算.Hetero-HPL 对原始 HPL 算法做出的改动可归纳为数据映射及操作映射两个方面,分别对应于如图 4 所示的 MD_Mapping 框架及 MD_Primitive 算法库.它们将在本节中分别予以介绍.

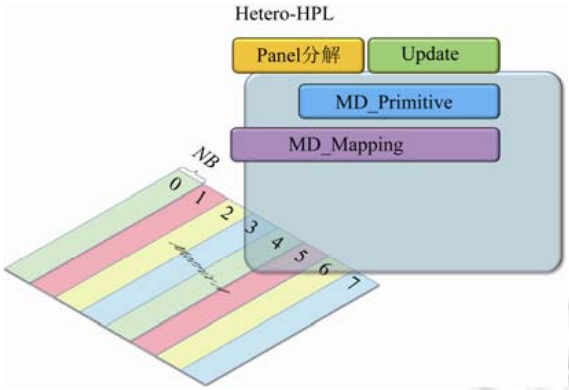

Fig.4 Architecture of Hetero-HPL

图 4 Hetero-HPL 总体结构

3.1 Hetero-HPL数据映射

在数据映射部分,Hetero-HPL 要解决如下 3 个问题:(1) 确保 Update 中 BLAS 3 函数完全由协处理器(组)完成,以接近计算平台的理论性能峰值;(2) 各个协处理器间计算相互独立,以避免冗余的跨协处理器的数据交换及相互依赖;(3) 在每一次迭代过程中,各个协处理器间应保持负载均衡.为同时解决这 3 个问题,我们为稠密矩阵在协处理器间进行数据划分提供一套框架,名为 Matrix-Device_Mapping(简称 MD_Mapping),提供高层接口便于开发者在逻辑上访问任意子矩阵,屏蔽了数据物理分布细节.

如图 5 所示,在每一个进程中 MD_Mapping 框架将 HPL 程序生成的初始伪随机矩阵划分到各个协处理器.由于在 Update 阶段,同一进程上的各个矩阵列间的操作是独立的,因此最自然的数据划分策略即为按矩阵列进行划分,以 *NB* 为宽度形成若干个列块.由于计算过程中,尾子矩阵的规模逐渐变小,若按照列方向线性划分并指派到各个协处理,则会导致协处理负载不均衡问题.因此,我们在 MD_Mapping 库中采用按列块循环卷帘的方式将诸多列块平均分配到各个协处理上.MD_Mapping 库以描述符的方式选取原始矩阵的任意子矩阵,即便这些子矩阵在物理上分布于不同的协处理器.运算围绕相关子矩阵实现.值得注意的是,假如我们对各个协处理器从 0 开始依次进行编号,并采用 MD_Mapping 中使用的列块循环卷帘数据排布方式,在同一列的不同进程上,原始矩阵中同一列元素所在的协处理器编号是相等的.这便于今后采用协处理间直接通信机制来优化 Hetero-HPL 程序性能.

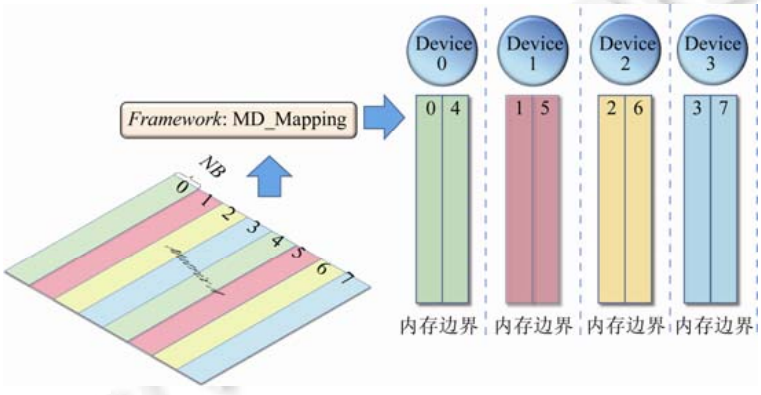

Fig.5 Matrix partition based on MD_Mapping framework

图 5 基于 MD_Mapping 框架的矩阵划分

3.2 Hetero-HPL操作映射

在 Update 阶段,HPL 算法主要涉及 3 种操作:(1) (分布式)行交换;(2) 上三角矩阵更新 dtrsm;(3) 尾子矩阵乘更新 dgemm.而在具有多设备的异构众核平台上,数据在 Host 及 Device 之间数据一致性也需要通过数据传递来实现.因此,我们在 MD_Mapping 框架上实现了 MD_Primitive 库.MD_Primitive 层以 MD_Mapping 提供的子矩阵描述符为操作对象,调用所有相关协处理器协同(并行)完成施加于目标子矩阵的操作,如数据传递,三角矩阵更新和矩阵乘更新等.另外,MD_Primitive 中的函数调用接口皆为异步接口,可以方便实施多操作重叠执行.在具备 MD_Mapping 框架及 MD_Primitive 库之后,Hetero-HPL 的大致程序结构如图 6 所示,其中,绿色字体代表同步调用,红色字体代表异步调用.

在整体算法开始之前,MD_Mapping 已经完成了对初始系统 $Ax=b$ 的映射,生成映射实例 MatrixMapping.由于 Hetero-HPL 主要对 HPL 算法的 Update 阶段进行了重构,因此我们围绕 Update 阶段进行阐述,其伪代码如图 6 左侧所示.通过 MatrixMapping 实例,程序获取两个子矩阵描述符(第 3 行~第 6 行),分别为 NextPanel(下一 Panel)及 TrailingMatrix(尾子矩阵).这两个子矩阵实例的实际数据分布细节被 MD_Mapping 框架屏蔽.MD_Mapping 为每个设备预留了数据缓存 L(可进一步划分为 $L1$ 和 $L2$)用来放置分解完毕的 Panel 数据.各个子矩阵实例可按需要分配属于自身的 Device 端数据缓存 U.数据缓存 L 及 U 将在接下来的步骤中使用,包括行交换(第 11 行及第 18 行),上三角矩阵更新(第 12 行及第 19 行)和矩阵乘更新(第 13 行及第 20 行).这些步骤由 MD_Primitive 库中函数实现.我们将施加于 NextPanel 的所有运算完成之后,开启异步传输(第 15 行),从对应的设备上将 NextPanel 的数据拷贝回 Host 端内存,进而完成下一次 Panel 分解操作.与此同时,对 TrailingMatrix 的上三角矩阵更新及矩阵乘更新可以通过异步接口与下一次 Panel 分解并行执行,如图 6 右侧所示.

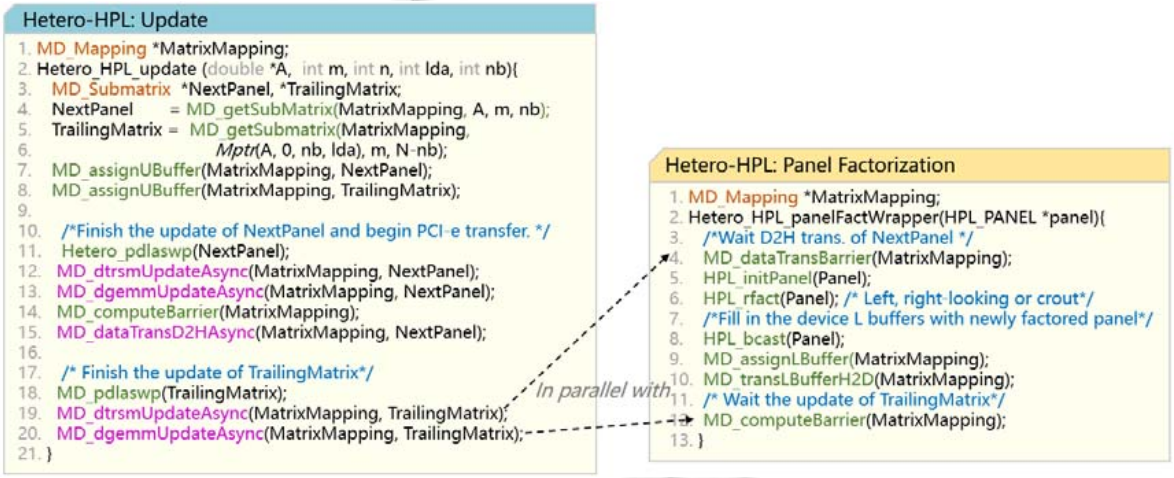

Fig.6 Pesudo-code of Hetero-HPL based on MD_Mapping and MD_Primitive

图 6 基于 MD_Mapping 及 MD_Primitive 实现的 Hetero-HPL 伪代码

4 面向异构平台的实现及优化

4.1 突破锁页内存分配大小限制

在现在主流异构平台上,Host 端内存和 Device 端协处理器内存的数据交换主要通过 PCI-e 总线完成.若要尽可能充分地利用 PCI-e 总线的带宽(约 32GB/s,全双工)则需要在 Host 端开辟锁页内存(pinned memory).锁页内存的最大特点是其不会被操作系统换出到硬盘交换区,以便于设备端建立物理地址映射,因此对协处理器来说可以通过 DMA 机制高速访问.相比于以常规方式申请的内存空间来说,锁页内存能够拥有 2~3 倍的

访问速率.然而,锁页内存一般为连续的物理内存空间,其申请受到单一 NUMA 节点上内存大小的限制.为确保数据在 Host 端及 Device 端对应,在拥有若干协处理平台上,单次申请的锁页内存容量很可能远小于所有设备内存的总和[10].因此,面向 Hetero-HPL,我们对原始矩阵建立软件缓存,采用若干锁页内存来替代非锁页内存.

如图 7 所示,以目标平台为例,假设原始矩阵大小为 64GB,在 Hetero-HPL 中,我们开辟两片(或多片)锁页内存区域(大小都为 32GB),然后采用同样的伪随机数生成算法直接初始化这两片内存区域.之后,在 LU 分解的过程中,Hetero-HPL 直接与这两片锁页内存完成多次矩阵 Panel 的相互交换.值得一提的是,原始的 64GB 非锁页内存区域在 Hetero-HPL 执行过程中并不需要真实存在,因此 Hetero-HPL 并不会带来冗余内存消耗.通过简单的地址变换,Hetero-HPL 也不会通过大量冗余操作来进行数据传输.在 Hetero-HPL 执行完成之后,我们获取解向量 $\boldsymbol{b}$ 并释放掉所有锁页内存.之后再由验证程序在 64GB 非锁页内存中重建整个线性系统,进而完成结果验证.

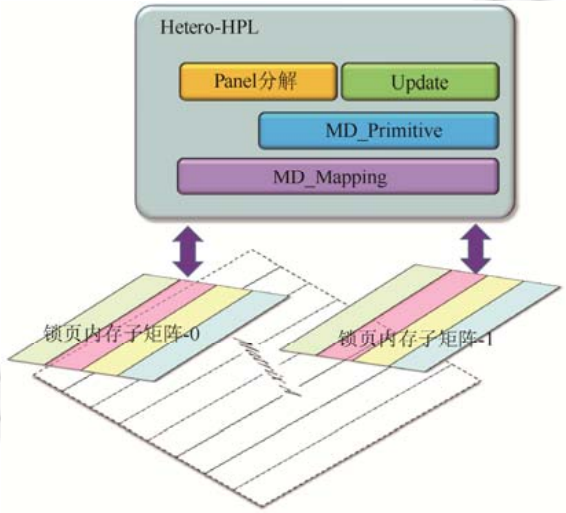

Fig.7 Original un-pinned memory is replaced by two pinned memory regions

图 7 采用两片锁页内存区域替代原始非锁页内存

4.2 同一协处理器上相同操作的归并

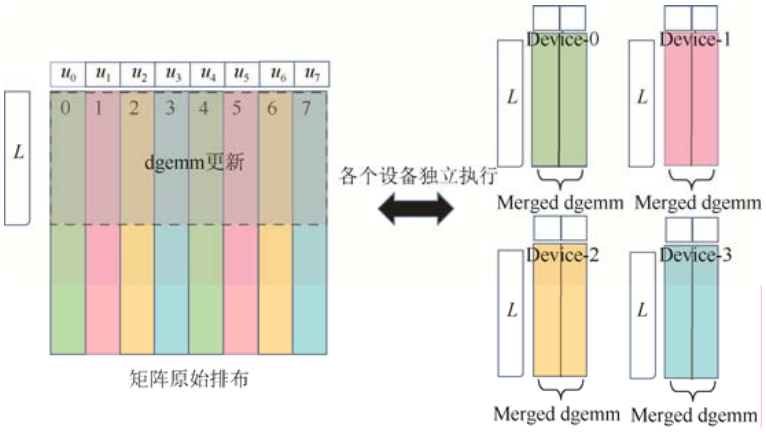

Fig.8 Merge of the same operation on each co-processor

图 8 协处理器上同种操作的归并

由于我们采用了按列块循环卷帘的方式在各个协处理器上分配矩阵数据,而每一列块的列数仅为 *NB*(通常为 128,256,512 等),因此施加在某一列块上的操作(如矩阵乘 dgemm 函数)可能会由于其数据规模过小而无法充分发挥协处理器的大规模并行计算能力.针对 HPL 算法程序,由于各个列块间计算相互独立,我们可以利用矩阵不同列间计算的结合律,让位于同一协处理的所有列块同时执行相同操作,即便这些列块在逻辑上并不相连.如图 8 虚线

框所示的子矩阵,其由多个列块构成.在每一个协处理器上,我们只需调用一次 dgemm 函数就可以完成该协处理器上所有列块的更新.类似地,同一进程中的行交换,dtrsm 等操作也可以进行归并,故不再赘述.

4.3 Panel分解与协处理器组异步并行

Panel 分解过程的各个步骤由于计算规模小,调用频繁而不利于使用协处理器进行加速.因此,在当前的 Hetero-HPL 版本中,我们将 Panel 分解放置在 CPU 端执行.如前文所述,相对于进程与协处理一一对应的方案而言,在 Hetero-HPL 中 Panel 分解操作将有条件利用更多 CPU 核心来完成计算.在 HPL 算法中下一次迭代步的 Panel 分解可以和当前次迭代的尾子矩阵 dgemm 更新并行执行.我们结合 MD_Primitive 库中的异步函数接口,可以很轻易地完成上述两个关键步骤的异步并行执行.

4.4 相关Kernel的优化实现.

由于待分解的矩阵被置于协处理器内存中,因此我们需要手工编写协处理器代码以实施相应运算.对于 BLAS 函数,我们可以直接调用 RocBLAS 库.而对于诸如行交换,矩阵转置等操作则需要我们手工编写计算 Kernel 完成.在此过程中,我们一方面要利用协处理器的大规模并行能力,另一方面我们也需要利用计算的可结合性,尽量在一次 Kernel 启动中完成同一协处理器上所有数据的计算.

5 实 验

5.1 实验平台简介

Hetero-HPL 面向当前主流的多协处理器架构构建,因此我们采用一个多核 CPU+多个 GPU 的典型异构系统作为实验平台.实验平台具有 Host 端及 Device 端.Host 端采用 4 路 8 核 Intel 指令集架构.与每路 CPU 核心邻近的内存大小为 32GB,系统共计 128GB 内存空间.Device 端为 4 个类 GPGPU 设备,每块类 GPGPU 拥有 64 个流处理器,16GB 设备内存.实验平台 Host 端支持 Posix 标准.因此可以采用常用的并行编程模型如 MPI、OpenMP 等.而对于 Device 端的类 GPGPU 设备,实验平台提供基础并行编程环境 Hip,并拥有开源基础并行算法库 Rocm[10].类似于 NVIDIA 公司推出的通用显卡计算编程环境 CUDA 及其加速算法库 CUDA Toolkit[11],实验平台能够支持手工编写并编译适用于 Device 端运行的计算 kernel,还可以直接利用 Rocm 算法库中的高性能函数实现并优化程序.另外,HPL 程序的高效执行离不开底层高效执行的基础线性代数库 BLAS.在 Host 端,实验平台采用 BLIS[4]数学库,其 dgemm 执行效率可以高达 CPU 理论峰值性能的 98%以上.而在 Device 端,我们同样可以利用 Rocm 算法库中的 RocBLAS 数学库模块,其 dgemm 性能约为单块协处理器理论性能的 84%以上.

5.2 单节点实验结果

Hetero-HPL 可以支持所有 128 倍数的 *NB* 取值.通过实验我们发现 *NB*=256 能够取得较优的总体性能.另外在 *NB*=256 时,Device 端矩阵乘法效率大约为机器峰值性能的 84%.我们选取 31 个 CPU 核心参与 Panel 分解计算,也能获得较优的总体性能.

图 9 展示了 Hetero-HPL 在单节点单进程情况下的运行效率.随着矩阵规模的扩大,执行效率呈现大幅度上升趋势.在矩阵阶数达到 88 320(4 个协处理器设备内存占用率达到 95.5%)时,Hetero-HPL 的运行效率达到了实验平台峰值性能的 76.53%,同时也达到了 dgemm 性能的 91.11%,因此可见 Hetero-HPL 在单节点单进程情况下能够取得较优的性能.

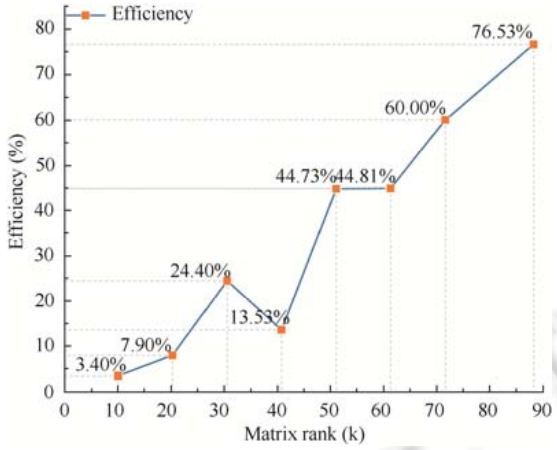

Fig.9 Efficiency trend of single process Hetero-HPL with respect to matrix rank

图 9 单进程 Hetero-HPL 执行效率随着矩阵阶数变化趋势

5.3 多节点实验结果

图 10 展示了 Hetero-HPL 在 4~256 个节点的运行性能.实验中每个进程的 Device 端内存使用量达到 96.0%.我们的工作展示了基于单进程控制多协处理器技术的 HPL 算法在分布式环境下的测试结果.但意外的是,Hetero-HPL 虽然在算法层面并没有引入多余的数据传输量,但是程序的性能随着进程数的增加而大幅度降低.由于 HPL 算法本身具有很好的可扩展性,而在实际测量中我们发现通信所耗费的时间约为总体计算时间的 25%~30%,因此我们认为通信效率成为制约 Hetero-HPL 在分布式环境下扩展性的一个重要方面.

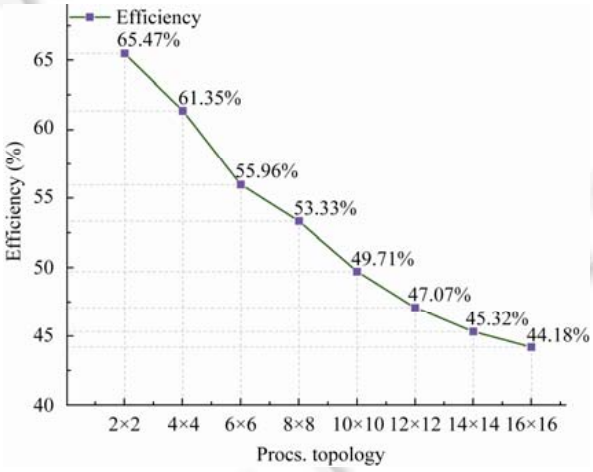

Fig.10 Scalability of Hetero-HPL on 4~256 nodes

图 10 Hetero-HPL 在 4~256 个节点可扩展性

Hetero-HPL 通信开销表现在 3 个方面.第一,PCI-e 总线利用率不高.以 Panel 分解相关的节点内传输为例,由于目前采用 *NB* 为列数按列方向划分矩阵,使得任意一个 Panel 从 Device 端拷贝到 Host 端仅能采用一路 PCI-e 总线.若使用 *NB*/*D*(*D* 为设备数量)为列数进行矩阵划分并在设备间卷帘排布,则可以确保任意一个 Panel 的数据均匀分布于所有设备,上述传输可以使用所有 PCI-e 总线并行完成.另外第 4.2 节所述的操作归并技术亦可在更细粒度的数据划分方案上使用,可确保设备端的计算性能.第二,Panel 分解阶段并未实现计算通信重叠.我们认

为能够利用 Panel 分解过程中 BLAS 3 级函数来掩盖部分 Panel 相关的数据传输.第三,进程间的 MPI 通信效率有待考证.由于 Hetero-HPL 不能直接控制底层网络资源(如多块网卡),因此如何配置 MPI 使其针对 Hetero-HPL 发挥最优性能也是一个值得深入研究的问题.

6 相关工作

自从 HPL 程序被提出并作为高性能计算评测标准,对其研究和改进及在不同平台上的并行方案、优化方法及计算模型研究就未曾中断过.从早期的 Intel Paragon 平台上的实现与测试[12],到现今世界最快的超级计算机上的优化与开拓[13],对 HPL 算法及优化的研究一直在影响着计算机软硬件设计与发展.异构平台上的 HPL 一般是主要由速度较快的加速器来完成矩阵更新操作,CPU 负责通信及 Panel 分解[6,7].Wang 等为 GPU+多核 CPU 平台设计了分阶段动态任务划分的方法及软件流水线,以充分利用计算和通信资源[8].Heinecke 等人针对带有 MIC 加速卡的平台,做了细粒度多层次的任务划分,从而实现对各部件的充分合理使用[9].Gan 等面向 HPL 在国产加速器 China accelerator 上对 dgemm 进行了优化,并使用静态与动态相结合的调度方式来协调 CPU 与加速器之间的工作[3].对多 GPU 平台上的 HPL 研究,陈任之等人工作采用了与本文工作类似的单进程控制多协处理器案[14].但是由于其并没有将尾子矩阵驻留于 Device 端,因此数据在 Host 端和 Device 端的来回传输对整体性能产生了负面影响.相似的方案也被 AMD 公司采用,为 HPL 中的 dgemm 实现了单进程多设备版本[11].与上述两个方案不同的是 Hetero-HPL 在多协处理器环境中也实现了全局数据的驻留,根本上避免了尾子矩阵的搬移.Jia 等人提出了与 Hetero-HPL 类似的解决方案,确保了数据多设备端的驻留并采用了按列块循环卷帘数据结构确保数据负载均衡[6].但值得指出的是,Jia 等人没有考虑到锁页内存分配大小限制小于 Device 端所有设备内存总量这一情况.在如本文所述的平台上,该方案只能处理较小阶数的矩阵,因而无法充分反映平台性能,因此其单节点性能仅达到机器峰值性能的 46.06%.另外,由于其方案采用了激进的动态调度策略,其复杂性也进一步降低了该方案在多进程环境中的可行性.在以上两方面,本文工作更为深入.

7 总结与未来的工作

面向当今主流的异构高性能计算平台,本文提出了 Hetero-HPL 测试程序.整体上,Hetero-HPL 所依赖的数据对协处理器的映射规则,并行策略等并不依赖于具体的硬件架构及基础软件,因此有能力在现有多种主流众核架构上进行部署.Hetero-HPL 还从根本上突破了进程数量与协处理器数量必须相等的限制,更进一步提高了其实用性.在执行过程中,单个进程携带多个协处理器的配置有利于在 Panel 分解和分布式行交换过程中利用更多了的计算资源,有利于执行效率的提高.Hetero-HPL 基于 MD_Mapping 框架和 MD_Primitive 库,使占主要计算量的尾子矩阵更新步骤完全利用多个协处理器提供的大规模并行计算能力,同时也避免了同一进程中多个协处理器间相互数据交换并确保各协处理器间计算量的均衡.面向多设备异构平台,我们突破了锁页内存分配限制,利用多块锁页内存在逻辑上替代原始非锁页内存,使 Hetero-HPL Device 端与 Host 端的内存交换仅发生在锁页内存中,确保了执行的高效.在本文中,我们面向多设备异构系统尝试了一些基础但必要的优化手段,其中包括 Panel 分解与尾子矩阵 dgemm 更新的异步并行,同一协处理器上相同计算的归并及行交换相关操作在协处理器端的并行化实现.实验结果表明,在单节点条件下 Hetero-HPL 达到了实验平台峰值性能的 76.53%(dgemm 实测性能的 91.1%).Hetero-HPL 还展示了基于单进程控制多设备技术的 HPL 算法在多节点分布式环境下的可行性.但目前由于网络通信及 PCI-e 数据交换的开销比例过大,程序执行效率还需进一步提高,这是我们将要深入研究的一个重要方面.

基于 Hetero-HPL,我们未来将主要开展两方面的工作.一方面,进一步在目标平台上深入优化 Hetero-HPL 程序.其中包括深入挖掘 Hetero-HPL 程序中的并行性,使更多关键步骤,如 Panel 在 Host 端及 Device 端的来回传递、跨进程的广播和分布式行交换等得以并行执行;在此基础上,分析各个步骤在各次迭代中的时间占比,基于动态信息反馈安排不同的异步并行方案.另一方面,我们可以在若干不同的平台上实现 Hetero-HPL,梳理制约 Hetero-HPL 性能的一般性因素,为 Hetero-HPL 的性能表现建立数学模型.

References:

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

[2] TOP-500 Official website. 2021. http://www.top500.org

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

[4] Van Zee FG, Van De Geijn RA. BLIS: A framework for rapidly instantiating BLAS functionality. ACM Trans. on Mathematical Software, 2013,41(3):1−33.

[5] Greer B, Henry G. High performance software on Intel Pentium Pro processors or micro-ops to TeraFLOPS. In: Proc. of the Supercomputing 1997 Conf. San Jose, 1997. 1−13. [doi: 10.1145/509593.509639]

[6] Jia Y, Luszczek P, Dongarra J. Multi-GPU implementation of LU factorization. In: Proc. of the Int'l Conf. on Computational Science, 2012. 106−115.

[7] 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(3-4):153−164.

[8] 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. [doi: 10.1007/s11390-011-0184-1]

[9] 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. 2013. [doi:10.1109/ipdps.2013.113]

[10] Fatica M. Accelerating Linpack with CUDA on heterogenous clusters. In: Proc. of the 2nd Workshop on General Purpose Processing on Graphics Processing Units. ACM, 2009. 46−51.

[11] Bach M, Rohr D. Scaling DGEMM to multiple Cayman GPUs and Interlagos many-core CPUs for HPL. 2011. http://developer.amd.com/wordpress/media/2013/06/2909_1_final.pdf

[12] Womble D, Greenberg D, Wheat S, Riesen R. LU factorization and the LINPACK benchmark on the Intel Paragon. Sandia Technical Report, Sandia National Laboratories, 1994.

[13] Offical website. 2021. https://www.olcf.ornl.gov/summit/

[14] Chen RZ, Huang LB, Chen XH, Wang ZY. Optimizing HPL benchmark on multi-GPU clusters. Computer Science, 2013,40(3): 107−110 (in Chinese with English abstract).

附中文参考文献:

[14] 陈任之,黄立波,陈琐颢,王志英.单节点多 GPU 集群下 HPL 动态负载均衡优化.计算机科学,2013,40(3):107−110.

孙乔(1989－),男,博士,高级工程师,主要研究领域为并行编程模型,并行算法.

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.

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.

赵玉文(1987－),女,博士生,助理研究员,CCF 专业会员,主要研究领域为高性能计算.