

数组维度类型程序设计方法及高性能 FFT 实现*

崔翔^{1,2,4}, 李晓雯³, 陈一峯^{1,2}

¹(高可信软件技术教育部重点实验室(北京大学), 北京 100871)

²(数学工程与先进计算国家重点实验室, 江苏 无锡 214125)

³(解放军防空兵学院, 河南 郑州 450000)

⁴(河南大学 计算机与信息工程学院, 河南 开封 475000)

通讯作者: 崔翔, E-mail: cuixiang08@pku.edu.cn



摘要: 异构集群多层次异构存储的特点, 决定了在其上进行计算时, 数据需要进行更多维度的划分. 现有集群程序设计语言缺乏对多维数组传输和转置的统一表示机制. 介绍多维数组维度转置的表示方法和课题组实现的 Parray 语言, 可以对异构集群复杂数据维度变换的数据操作进行清晰表示. 同时介绍基于数组维度类型程序设计方法和 Parray 语言实现的天河 1A 系统上的大规模 3 维 FFT, 该算法代码实现简洁, 同时得到了良好的性能和可延展性.

关键词: 异构集群; 集群程序设计语言; 离散傅里叶变换(FFT)

中图法分类号: TP311

中文引用格式: 崔翔, 李晓雯, 陈一峯. 数组维度类型程序设计方法及高性能 FFT 实现. 软件学报, 2015, 26(12): 3104-3116. <http://www.jos.org.cn/1000-9825/4801.htm>

英文引用格式: Cui X, Li XW, Chen YF. Programming method of dimensional array types and high performance FFT implementation. Ruan Jian Xue Bao/Journal of Software, 2015, 26(12): 3104-3116 (in Chinese). <http://www.jos.org.cn/1000-9825/4801.htm>

Programming Method of Dimensional Array Types and High Performance FFT Implementation

CUI Xiang^{1,2,4}, LI Xiao-Wen³, CHEN Yi-Feng^{1,2}

¹(Key Laboratory of High Confidence Software Technologies of Ministry of Education (Peking University), Beijing 100871, China)

²(State Key Laboratory of Mathematical Engineering and Advanced Computing, Wuxi 214125)

³(Air Defense Forces Academy, Zhengzhou 450000, China)

⁴(College of Computer & Information Engineering, He'nan University, Kaifeng 475000, China)

Abstract: Because a heterogeneous cluster relies on a heterogeneous storage system, the data needs to be divided in a multi-dimensionally manner when doing computation on it. Current cluster-level programming languages have no unified representation mechanisms of transmission and transposition of multi-dimensional arrays. This article describes the programming method based on multi-dimensional array types and the Parray language, which can be used to represent the complex multi-dimensional data transposition on heterogeneous clusters in a clear way. A large-scale three-dimensional FFT implementation on Tianhe 1A based on the array type

* 基金项目: 国家自然科学基金(61240045, 61170053, 61432018, 61379048); 国家高技术研究发展计划(863)(2012AA010902, 2012AA010903); 中国博士后科学基金(2013M540821); 数学工程与先进计算国家重点实验室开放课题(2013A12); 河南省教育厅科学技术研究重点项目(13A520065)

Foundation item: National Natural Science Foundation of China (61240045, 61170053, 61432018, 61379048); National High-Tech R&D Program of China (863 Program) (2012AA010902, 2012AA010903); China Postdoctoral Science Foundation (2013M540821); the State Key Laboratory of Mathematical Engineering and Advanced Computing (2013A12); the Science and Technology Key Project of Education Department of He'nan Province (13A520065)

收稿时间: 2013-11-28; 定稿时间: 2014-12-09

programming method and Parray is also introduced. The final code is very simple but gets a good performance and scalability at the same time.

Key words: heterogeneous clusters; cluster programming language; discrete Fourier transform (FFT)

超级计算机的架构呈现出异构化的发展方向。目前,全球运算速度最快的天河-2 超级计算机使用 Intel 公司的 MIC 协处理器进行加速^[1],全球运算速度第二的 Titan 超级计算机使用 Nvidia Tesla K20X GPU 进行加速,具有异构加速计算部件的 workstation 集群,简称异构集群,是指在工作站集群的每个节点上不仅仅使用传统 CPU 进行计算的集群架构。异构加速计算部件也日益呈现出多样性,我国也有自行设计实现的众核处理器。异构加速计算部件对于特定计算任务表现出良好的性能^[2,3]。

异构集群的多层次异构存储特点,决定了在异构集群进行计算时,数据需要进行更多维度的划分;同时,数据需要在不同的存储结构上进行复杂的传输和维度变换。以 GPU 加速集群的 3 维 FFT 程序为例,数据维度的划分达到 9 维之多,而且需要对数据进行包含节点维度和节点内 GPU 维度(甚至 GPU 内单核分布维度)的多维转置^[4]。

现有的程序设计语言缺乏对多维数组传输和转置的统一表示机制,该表示机制应该对不同层次异构存储中的数据分布和传输转置进行统一精确的描述。通过对异构集群多层次存储结构数据维度变换表示方法的研究,包括在表示法中引入数据维度与异构集群体系结构层次的对应关系(如分布到集群不同节点的维度、分布到一个节点不同 GPU 的维度、国产众核处理器中分布到一个处理器不同从核的维度),可以得到计算过程中数据在不同存储部件分布、传输和转置的精确描述。同时,通过数组维度类型程序设计方法,程序员可以在代码书写阶段简便清晰地表示异构集群中复杂的数据传输和转置操作。

本文以异构集群的高性能 3 维 FFT 为例,介绍基于 Parray 语言的数组维度类型程序设计方法。在异构处理器 FFT 的研究方面,Nvidia 公司有针对单个 GPU 的数学库 CUBLAS 和 CUFFT,也正在研制针对单机上多个 GPU 的数学库;很多研究人员进行了单 GPU 上 FFT 算法的研究^[2,5-14]。在发布 MIC 协处理器的同时,Intel 公司也推出了 MIC 协处理器上的 MKL 数学库。现有的传统集群上的商业版 FFT 数学库主要有 FFTW^[15]和 Intel MKL^[16]。FFTW 是使用最为广泛的 FFT 库,其基于 MPI 的 3 维 FFT 并没有考虑计算和通信的重叠优化问题;P3DFFT^[17,18]及其他许多研究^[19-22]通过对数据进行节点间高维分布,以提高 3 维 FFT 的延展性,但没有考虑计算和通信的重叠优化问题;2DECOMP&FFT^[23]和 Kandalla 等人^[24]通过优化多组输入数组间的计算通信重叠问题,实现对批量数据 3 维 FFT 计算的性能优化。针对一些科学计算问题^[25,26]中需要对一个数据数组进行多次 3 维 FFT 计算的应用场景,Song 等人^[27]通过优化一次 3 维 FFT 内的计算通信重叠问题来提升性能。Bell 等人^[28]基于 UPC 和支持异步通信的硬件,实现了一次 3 维 FFT 内的计算通信重叠优化。Fang 等人^[29]使用特定的通信 API,在多维网络结构中实现了计算通信重叠的 3 维 FFT。Doi 等人^[30]考虑非对称网状网络的拓扑结构,合理调度远距离进程之间的通信,在多核节点上使用不同的线程,实现通信和计算的重叠。近年来,基于异构集群的 3 维 FFT 算法的研究也很活跃^[31,32],我们也发表过自己的工作^[4],对于 GPU 异构集群,在 GPU 设备存储器、主存、Infiniband 网络之间协调并发数据通信实现高性能 3 维 FFT。在 HPCC 的集群 FFT 测试^[33]中,目前最快的集群 FFT 是 IBM Blue Gene,达到 226.10TFlop/s。

本文第 1 节介绍现有的集群编程语言,它们多为 PGAS(partitioned global address space)语言,现有语言从概念上对数据的存储层次进行隐藏,且缺乏对新兴异构加速计算部件的编程支持。第 2 节介绍多维数组维度转置的表示方法,可以对集群 FFT 这种需要复杂维度变换的数据操作进行清晰表示。第 3 节介绍课题组实现的 Parray 语言中多维分布数组的表示和传输方法。第 4 节以在天河 1A 系统上实现的大规模集群 FFT 为例,说明 Parray 语言进行复杂维度数据程序设计时代码书写的简洁性。该 FFT 最大运行到 14336×14336×14336 单精度复数的规模,共使用 7 168 个节点,这是目前异构架构下节点规模最大的 3 维 FFT 运算。同时,该算法取得了超越 Intel MKL 数学库的良好性能。

1 相关工作

现有的集群并行程序设计语言主要使用于传统集群.表 1 对已有集群编程语言的特性进行了比较.第 1 列根据语言的实现方式进行了分类:(1) 以库的方式实现数据通信和共享的操作,而其他操作以传统串行语言元素表示;(2) 以语言结构或原语显式或隐式的表示并行语义.表中第 2 列按照并行程序的每一个执行元素可见的地址空间对语言进行了分类:在 MPI 中,一个进程只能直接访问其所在节点的局部数据,而其他所有的编程模型都允许线程访问位于任何节点的全局数据.表中第 3 列描述了不同语言的编程控制能力.

Table 1 Comparison of current cluster programming languages

表 1 已有集群编程语言的特性比较

语言	实现		地址空间		控制		
	语言	库	全局	局部	集群	多核	众核
Chapel	✓		✓		✓	✓	
CAF	✓		✓		✓		
HPF	✓		✓		✓		
HTA		✓	✓		✓	✓	
MPI/P VM		✓		✓	✓		
Titanium	✓		✓		✓		
UPC	✓		✓		✓		
X10	✓		✓		✓	✓	
ZPL	✓		✓		✓		

这些集群并行程序设计语言都从一定程度上支持所谓数组或数据域(domain)的操作.通常的支持包括分块、循环的数据分布方式和数据复制等.CAF 通过所谓 co-array 的数组维度,表示一个 SPMD 程序所操作的多个数据实例;Titanium 在 Java 中增加了新的特性以支持多维数组、子数组的操作;ZPL 使用一系列的数组运算符表示不同的数组操作模式,包括转置、广播、规约等;UPC 中可以声明共享数组变量,使数据分布在不同的程序实例上;X10 也支持可以分布在集群节点上的多维数组操作.

一类集群并行程序设计语言利用消息传递实现多个进程间的数据通信和相互协作,如 MPI;另一大类集群编程语言为 PGAS 语言.PGAS 将整个集群的存储器视为统一编址的空间,支持像共享存储一样的变量访问并忽略通信问题,而希望程序员来考虑数据局部性的问题.在 PGAS 中,仅仅访问一个整型变量,由于其存储位置不在本地节点,便可以引发通信,数据传输粒度无法控制.此外,由于对共享存储访问操作的原子性,难以实现数据传输和计算之间的并行.总之,PGAS 语言的理念是对集群体系结构的抽象、隐藏、包装,而异构集群高性能编程的特点则要求对其体系结构特征进行暴露、需要程序员了解它并有针对性的编程.PGAS 语言不适合异构集群高性能程序设计的要求.

2 多维数组维度变换

在多维数组维度表示法中,对多维嵌套数组的不同维度进行命名,并对分布维度进行符号标示.这样,包括以集群 FFT 为例的复杂的数据转置变换可以得到清晰的描述.而在第 3 节中,通过使用 Parray 语言,可以对多维数组的逻辑维度进行直接编程表示,从而实现多维数组维度表示法和 Parray 代码实现概念上的无缝结合.

2.1 多维数组维度表示方法

一个逻辑上 8 行 8 列的数组,在内存中可以有多种分布方式.

图 1(a)表示了一个行优先(维度 x 的数据在内存中是连续存储的)的数组,该数组被表示为 $\frac{y}{[8]} \frac{x}{[8]}$.经过一个

本地转置,将维度 x 和维度 y 进行调转之后,该数组变成如图 1(b)所示的分布,记为 $\frac{x}{[8]} \frac{y}{[8]}$.此时,该数组以列优先的方式存储.在这种表示法中,横线之下方括号中的数字表示维度的大小,横线之上为维度的名称.

如果图 1(a)所示的数组被按照 y 维度进行划分,分布到集群中 4 个不同的节点上,如图 1(c)所示,则原有的 y 维度分裂为 4×2 ,此时,该数组记为 $\left[\begin{matrix} \hat{y}_1 & y_0 \\ [4] & [2] \end{matrix} \right] \frac{x}{[8]}$. 其中,维度 y_1 之上的^符号表示该维度为分布在集群上的维度,维度 y_1 和 y_0 之外的方括号表示 y_1 和 y_0 为嵌套的维度,它们由之前的 y 维度分裂而来.对图 1(b)中的数组进行 x 维度的划分并分布到 4 个不同的节点上,分布图如图 1(d)所示,记为 $\left[\begin{matrix} \hat{x}_1 & x_0 \\ [4] & [2] \end{matrix} \right] \frac{y}{[8]}$.

这种表示方法同样可以表示多维的划分.例如,将图 1(a)中的数组按照 x 和 y 维度进行划分,并且分布到 2×2 的节点阵列上,记为 $\left[\begin{matrix} \hat{y}_1 & y_0 \\ [4] & [2] \end{matrix} \right] \left[\begin{matrix} \hat{x}_1 & x_0 \\ [4] & [2] \end{matrix} \right]$. 这里, y_1 维度和 x_1 维度之上的不同的符号对应于节点阵列不同的行维度和列维度.通过这种表示法可以清楚地看出: $\frac{\hat{y}_1 \hat{x}_1}{[4][4]}$ 为节点阵列的数组,而 $\frac{y_0 x_0}{[2][2]}$ 为每个节点上数据块的数组.

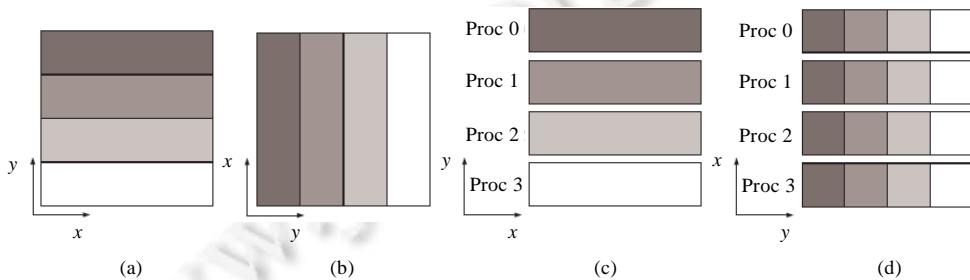


Fig.1 Array layout in memory

图 1 数组在内存中的分布

2.2 传统集群3维FFT维度变换

传统集群的 3 维 FFT 可以用此维度表示法进行清晰的描述.如图 2(a)所示,为 $8 \times 8 \times 8$ 的 3 维立方体数据,其中, z 维度为最密集的,而 x 维度为最稀疏的,此时,数组可以记为 $\frac{x \ y \ z}{[8][8][8]}$. 将该数组分布到 4 个节点的集群上去,一种方式是进行 1 维的分割,即将该数组按照 x 维度的方向进行分割,如图 2(b)所示.此时,该数组记为 $\left[\begin{matrix} \hat{x}_1 & x_0 \\ [4] & [2] \end{matrix} \right] \frac{y \ z}{[8][8]}$,这意味着数据分布在 4 个节点上,每个节点有 $2(x_0$ 维度)个 8×8 (y 和 z 维度)的正方形数据.

进行 3 维 FFT 计算的步骤如下:

1. 在 $4(x_1$ 维度)个节点上分别并行进行 $2(x_0$ 维度)次 2 维 8×8 的 FFT,将 y 和 z 维度方向的 1 维 FFT 计算完成;
2. 需要进行 x 维度方向的 FFT,由于 x 维度方向的数据分布在不同的节点上,因此需要进行通信.将数据再进行 y 维度方向的分割,如图 2(b)所示.此时,该数组记为 $\left[\begin{matrix} \hat{x}_1 & x_0 \\ [4] & [2] \end{matrix} \right] \left[\begin{matrix} y_1 & y_0 \\ [4] & [2] \end{matrix} \right] \frac{z}{[8]}$. 通过进行 2 次(x_0 维度)的分组 AllToAll 通信,即可将 x_1 维度和 y_1 维度进行对调,数据的分布变为 $\frac{\hat{y}_1 \ x_0 \ x_1 \ y_0 \ z}{[4][2][4][2][8]}$;
3. 数据已经按照 y_1 维度分布在 4 个节点上,但 x 维度的数据在内存中仍不连续.为使 x 维度的数据连续以保证 cache 访问的友好性,亦可在各个节点并行进行本地的数据转置,将其变为 $\frac{\hat{y}_1 \ y_0 \ z}{[4][2][8]} \left[\begin{matrix} x_1 & x_0 \\ [4] & [2] \end{matrix} \right]$, 即 $\frac{\hat{y}_1 \ y_0 \ z \ x}{[4][2][8][8]}$ (注意:此转置需要额外的内存缓冲区空间);
4. 在完成了对 x 维度数据的 1 维 FFT 之后,可以将整个步骤反序进行,把数据元素恢复到初始的位置.

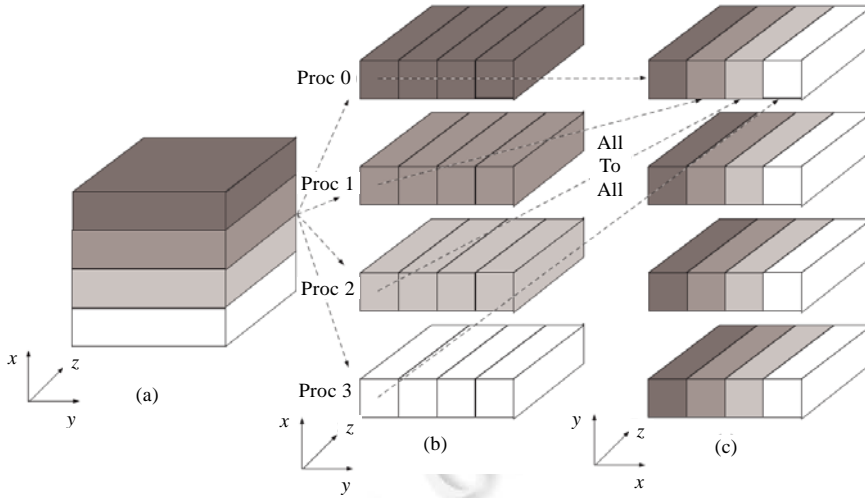


Fig.2 3D FFT on traditional clusters
图 2 传统集群的 3 维 FFT

传统集群 3 维 FFT 的另一种分布数据的方法是,按照 2 个维度对数据进行分布.对 $8 \times 8 \times 8$ 的 3 维立方体数据,按照 x 维度和 y 维度进行划分,分布到 4×4 的节点阵列上,记为 $\begin{bmatrix} \hat{x}_1 & x_0 \\ [4] & [2] \end{bmatrix} \begin{bmatrix} \hat{y}_1 & y_0 \\ [4] & [2] \end{bmatrix} \frac{z}{[8]}$.进行 3 维 FFT 计算的步骤如下:

1. z 维度方向的数据都在同一个节点本地,16 个节点并行计算 $2(x_0 \text{ 维度}) \times 2(y_0 \text{ 维度})$ 个 z 维度方向的 1 维 FFT;
2. 为了通信的需要,将 z 维度做进一步的分裂,记为 $\begin{bmatrix} \hat{x}_1 & x_0 \\ [4] & [2] \end{bmatrix} \begin{bmatrix} \hat{y}_1 & y_0 \\ [4] & [2] \end{bmatrix} \begin{bmatrix} \hat{z}_1 & z_0 \\ [4] & [2] \end{bmatrix}$.按照 x_1 维度,将节点阵列分为 4 组,每组 $4(y_1 \text{ 维度})$ 个节点,各组节点内做 $2(x_0 \text{ 维度}) \times 2(y_0 \text{ 维度})$ 次分组 AllToAll 通信,将 y_1 维度和 z_1 维度转置,得到 $\begin{bmatrix} \hat{x}_1 & x_0 \\ [4] & [2] \end{bmatrix} \begin{bmatrix} \hat{z}_1 & y_0 & y_1 & z_0 \\ [4] & [2] & [4] & [2] \end{bmatrix}$;
3. y 维度方向的数据都在同一个节点本地,16 个节点并行计算 $2(x_0 \text{ 维度}) \times 2(z_0 \text{ 维度})$ 个 y 维度方向的 1 维 FFT;
4. 按照 z_1 维度,将节点阵列分为 4 组,每组 $4(x_1 \text{ 维度})$ 个节点,各组节点内做 $2(x_0 \text{ 维度}) \times 2(y_0 \text{ 维度})$ 次 AllToAll 通信,将 x_1 维度和 y_1 维度转置,得到 $\begin{bmatrix} \hat{y}_1 & x_0 & \hat{z}_1 & y_0 & x_1 & z_0 \\ [4] & [2] & [4] & [2] & [4] & [2] \end{bmatrix}$;
5. 在完成了对 x 维度数据的 1 维 FFT 之后,可以将整个步骤反序进行,把数据元素恢复到初始的位置.

3 维度变换的 Parray 语言表示

在多维数组维度表示法中,对多维嵌套数组的不同维度进行命名.通过本节介绍的 Parray 语言^[34],可以在语言层面对多维数组的维度进行直接编程.

3.1 自然数组

Parray 语言中,自然数组是指数组数据元素的逻辑下标和物理偏移相一致的数组类型.如图 3 所示,为 Parray 的自然数组示例代码.

在 Parray 代码中,与 Parray 有关的成分通常为以 \$ 起始的一段代码.这样,Parray 编译器可以对该代码方便地

进行预处理,提取 `Parray` 有关的语言成分,并展开成为目标 C 语言代码.在该代码片段中,第 1 行首先声明一个数组类型 `A`,该类型为 `[8][8]` 维度的浮点数(可以视为 8 行 8 列的数组),且存储的物理位置为分页内存(`paged` 表示);该行代码只是声明一种类型,即,看待物理存储器中数据存储的一种方式,而并未分配其需要的真正的物理存储空间.而后,第 2 行代码使用 `Parray` 的 `create` 语句,声明一个浮点数的指针 `x`,按照 `A` 类型的需要分配了相应的物理存储空间,不分页内存中可以存储 `8×8` 个浮点数的空间,空间的起始地址存储在指针 `x` 中.第 3 行代码对 `x` 指向的数组进行初始化.第 4 行代码打印 `x[SA[5][2]]` 的元素值,其中 `SA[5][2]` 代表对应 `A` 类型的行和列维度(即 `A_0` 维度和 `A_1` 维度)值分别为 5 和 2 的元素的物理偏移值.最后,第 5 行代码释放 `x` 指针指向的对应类型 `A` 的物理存储空间.

```

1.  $parray {paged float[8][8]} A
2.  $create A(x)
3.  $for i::A(x) {(*x)=i;}
4.  printf("array access:%d\n",x[SA[5][2]]);
5.  $destroy A(x)

```

Fig.3 Natural array in Parray

图 3 Parray 自然数组代码示例

3.2 人工数组

在 `Parray` 中,数组的物理存储与逻辑视图在概念上进行分离:类型提供且仅提供对数据的逻辑视图,与数据的物理存储空间无关;数组维度变换和转置通过人工数组实现.人工数组是指含有一个或多个人工维度的数组类型,人工维度来自于对已有数组类型维度的引用.人工数组代码示例如图 4 所示.

```

1.  $parray {paged float[8][8]} A
2.  $parray {[#A_1][#A_0]} B
3.  $create A(x)
4.  x[SA[3][2]]= 123;
5.  printf("array access:%d\n",x[SB[2][3]]);
6.  $destroy A(x)

```

Fig.4 Artificial array in Parray

图 4 Parray 人工数组代码示例

图 4 的代码中,第 1 行声明一个 8 行 8 列的自然数组类型 `A`,第 2 行声明一个人工数组类型 `B`,其两个维度为 `#A_1` 和 `#A_0`.以 `#` 起始的维度代表该维度引用自其他数组类型的已有维度,并同时引用其维度数值和偏移函数.本例中,数组类型 `B` 的两个维度将 `A` 的两个维度进行了调转.第 4 行代码以类型 `A` 的视图对数组第 3 行第 2 列的元素赋值 123,第 5 行代码以类型 `B` 的视图取得数组第 2 行第 3 列的元素,即,同为 123.

3.3 线程数组和混合数组

`Parray` 中,使用 `parray` 命令同样可以声明线程数组.

如下代码声明一个 `Pthread` CPU 线程数组类型 `MYTA`,其维度为 `2×2`:

```
1.  $parray {pthd[2][2]} MYTA
```

如下代码声明一个 `CUDA` 线程数组类型 `MYCUDATA`,其维度为 `8×8×16×16`:

```
1.  $parray {cuda(float2*worki,float2*worko)[[128/16][128/16]][[16][16]]} MYCUDATA
```

如下代码声明一个 `MPI` 进程数组类型 `MYMPITA`,其维度为 2:

```
1.  $parray {mpi[2]} MYMPITA
```

`Parray` 中,线程数组的维度可以和数据数组的维度结合在一起,构成一种特殊的人工数组,称为混合数组.混合数组可以用来表示分布的数据.图 5 的代码中,第 3 行即声明了一个混合数组类型 `T`:它的 `T_0_0` 维度来自 `MPI` 进程数组类型 `MYTA`,而 `T_0_1` 和 `T_1` 维度来自于分页内存数组类型 `DT`.混合数组类型 `T` 表示的数据意义为(如

图 1(c)所示): 8×8 个整数构成的正方形数据,分布在 4 个 MPI 进程的不同节点的分页存储器上.

使用混合数组可以形象地表示分布的数据,无论数据是分布在多个 MPI 进程上,或者是分布在多个 Pthread 线程上.

1. `$parray {mpi[4]} MYTA`
2. `$parray {paged int[[4][2]][[8]]} DT`
3. `$parray {[[#MYTA][#DT_0_1]][#DT_1]} T`

Fig.5 Parray Mixed array in Parray

图 5 Parray 混合数组示例

将图 5 的代码稍作改写,将 *DT* 类型的 *DT_1* 维度分裂为 4×2 的维度得到:

1. `$parray {mpi[4]} MYTA`
2. `$parray {paged int[[4][2]][[4][2]]} DT`
3. `$parray {[[#MYTA][#DT_0_1]][#DT_1_0][#DT_1_1]} T`

将此分布存储的数组进行集群间的转置,由图 1(c)所示的分布变换为图 1(d)所示的分布,则对应的数据类型表示为(将#*MYTA* 和#*DT_1_0*,#*DT_0_1* 和#*DT_1_1* 维度进行了对换):

1. `$parray {[[#DT_1_0][#DT_1_1]][#MYTA][#DT_0_1]} T`

同理,可以得到集群 3 维 FFT 的数据维度分布及其变换的 Parray 表示.在下节中,将介绍混合数组类型可以用来进行分布数据的传输.

3.4 维度变换的数据传输

Parray 中,可以通过调用 `copy` 子程序完成集群节点上甚至节点间的数据传输和转置,其语法为

$$\$copy\ S(s)\ to\ T(t),$$

其中,参数 *s* 和 *S* 为传输起始的数据地址和数组类型,参数 *t* 和 *T* 为传输目标的数据地址和数组类型,起始类型 *S* 和目标类型 *T* 需要具有相同的维度树结构.`copy` 所完成的数据传输将根据起始类型 *S* 和目标类型 *T* 指示出的维度关系,将数据由起始地址 *s* 复制到目标地址 *t*.

图 6 的代码完成从不分页内存到 GPU 设备内存的连续数据传输.

1. `$parray {pinned float[8][8]} A`
2. `$parray {dmem float[8][8]} B`
3. `$create A(a), B(b)`
4. `$copy A(a) to B(b)`
5. `$destroy A(a), B(b)`

Fig.6 Continuous data copy in Parray

图 6 Parray 连续数据复制

图 6 中:第 1 行声明一个不分页内存中的自然数组类型 *A*;第 2 行声明一个和类型 *A* 具有同样维度结构的 GPU 设备内存中的自然数组类型 *B*;第 3 行代码声明两个指针并创建两种数组类型对应的存储空间;第 4 行代码调用 `copy` 子程序,将位于不分页内存中的指针 *a* 指向的类型为 *A* 的数组,传输到位于设备内存中的指针 *b* 指向的类型为 *B* 的数组.实际上,Parray 编译器判断出该数据传输为连续的,会将该 `copy` 翻译成 1 个 `cudaMemcpy` 语句,进行 64 个浮点数的连续数据复制.

图 7 的代码完成从不分页内存到 GPU 设备内存的转置数据传输,维度由 $\frac{y}{8} \times \frac{x}{8}$ 转置为 $\frac{x}{8} \times \frac{y}{8}$,如图 1(a)和图 1(b)所示.

```

1. $parray {pinned float[8][8]} A
2. $parray {dmem float[#A_1][#A_0]} B
3. $create A(a), B(b)
4. $copy A(a) to B(b)
5. $destroy A(a), B(b)

```

Fig.7 Transposed data copy in Parray

图 7 Parray 转置数据复制

在使用多 GPU 的计算中,经常会用到的一种数据传输模式如图 8 所示:1 个节点上安装有 2 个 GPU,分别被 2 个 Pthread 线程控制,在主存中有 4096×4096 个元素的数据,此时,需要将该正方形数据的前一半(前 2 048 行的数据)通过 PCI 总线传输到 0 号 GPU,将该正方形数据的后一半(后 2 048 行的数据)传输到 1 号 GPU.这样的程序通过现有的技术书写起来还是比较复杂的,但通过 Parray 的混合数组类型和数据传输机制,可以很容易地书写出这样的代码.

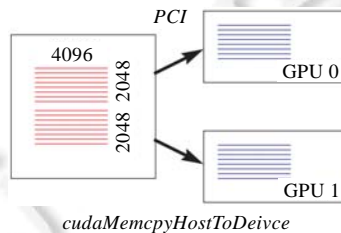


Fig.8 Distributed data copy to GPUs

图 8 GPU 分布数据传输

图 9 的代码完成图 8 所示意的数据传输:

- 第 4 行声明起始数据类型 S ,它是在分页存储器中的整数类型,维度为 $2 \times 2048 \times 4096$,对应到图 8 中左侧的来源数据格式;
- 第 3 行声明目标数据类型 T ,它的 T_0 维度为 $\#MYTA$,即 2 个 Pthread 线程, T_1 维度为 $\#DT$,即 2048×4096 半个正方形的数据,数据类型 T 对应到图 8 中右侧的目标数据格式;
- 第 9 行代码,main 主进程调用 $MYTA$ 线程数组,使得 2 个 Pthread 线程分别初始化自己的 GPU 和设备存储器;
- 第 12 行代码, $MYTA$ 线程数组使用 copy 子程序完成图 8 所示意的数据传输.

实际上,这个 copy 子程序最终会被转化为 2 个 Pthread 线程中的 `cudaMemcpyHostToDevice()` 函数调用.

```

1. $parray {pthd[2]} MYTA
2. $parray {dmem int[2048][4096]} DT
3. $parray {[[#MYTA]][#DT]} T
4. $parray {paged int[[2]][#DT]} S
5. int*host;
6.
7. $main{
8.   $create S(host)
9.   $for tid::MYTA{
10.    INIT_GPU(tid);
11.    $create DT(dev)
12.    $copy S(host) to T(dev)
13.   }
14. }

```

Fig.9 Distributed GPU data copy code in Parray

图 9 GPU 分布数据传输 Parray 代码

图 8 所示意的数据传输模式也可能发生在集群节点之间,如图 10 所示.此时,一个节点上有 4096×4096 个元

素的数据,需要将该正方形数据的前一半(前 2 048 行的数据)通过网络传输到 0 号进程,将该正方形数据的后一半(后 2 048 行的数据)传输到 1 号进程.这样的程序通过现有的技术需要书写调用 MPI 库的通信函数,但通过 Parray 的混合数组和数据传输机制,同样的思路即可以得到这样的代码.

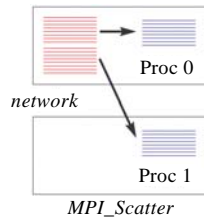


Fig.10 Distributed data copy of MPI

图 10 MPI 分布数据传输

图 11 的代码完成图 10 所示意的数据传输,与图 9 中的代码比较,其中需要进行的主要修改用粗斜体标出:第 1 行将 MYTA 线程数组类型由 Pthread 线程数组修改为 MPI 进程数组;第 2 行将 DT 数据类型由 GPU 设备存储器修改为分页存储器类型.可以看出,对于图 8 和图 10 所表示的数据传输模式,其 Parray 代码的书写是十分类似的.而实际上,这个 copy 子程序最终会被转化为 MPI 进程中的 MPI_Scatter()函数调用.

```

1.  Sparray {mpi[2]} MYTA
2.  Sparray {paged int[2048][4096]} DT
3.  Sparray {[[#MYTA]][#DT]} T
4.  Sparray {paged int[[2]][#DT]} S
5.
6.  $main{
7.    $create S(host)
8.    $for MYTA{
9.      $create DT(dev)
10.     $copy S(host) to T(dev)
11.    }
12.  }

```

Fig.11 Distributed data copy code of MPI in Parray

图 11 MPI 分布数据传输 Parray 代码

4 异构集群的 FFT 实现

天河 1A 是我国自主组建的大型 GPU 异构集群系统.该集群共有 8 100 个节点,每一个节点接有 2 个 6 核 Intel 处理器和 1 个 448 核 Tesla Fermi GPU.该集群采用定制的网络并使用胖树的交换结构,每一个节点具有 80Gbps 的理论带宽.

在天河 1A 上,使用 Parray 书写的 3 维 FFT 代码最大运行到 14336×14336×14336 单精度复数的规模,共使用 7 168 个节点,这是目前世界上最大规模的 3 维 FFT 运算.在此,以使用 4 096 个节点计算 8192×8192×8192 规模的数据为例,介绍 Parray 3 维 FFT 代码的设计与实现.

使用数组维度表示方法,初始数据记为

$$\frac{x \quad y \quad z}{[8192][8192][8192]}$$

将数组数据按照 x 维度分割为 4 096 份,分布到 4 096 个节点上,这样,每一个节点的数据为 2×8192×8192 个单精度复数.此时,数据记为

$$\frac{\hat{x}_1 \quad x_0 \quad y \quad z}{[4096][2][8192][8192]}$$

首先,每个 GPU 先进行 2(x_0 维度)个 2 维的 FFT,将 y 维度方向和 z 维度方向的数据进行计算.由于计算的时

间与数据传输的时间相比基本上可以忽略,在此直接使用 CUFFT 调用即可,无需使用已实现的更加优化的 GPU FFT 代码.

在进行 x 维度的 FFT 计算之前,需要进行转置,将 y 维度的数据做进一步的划分,得到:

$$\frac{\hat{x}_1 \quad x_0 \quad y_1 \quad y_0 \quad z}{[4096][2][4096][2][8192]} \dots$$

在集群上直接使用 copy 子程序进行通信,将数据分布变换为

$$\frac{\hat{y}_1 \quad y_0 \left[\begin{array}{c} x_1 \quad x_0 \\ [4096][2] \end{array} \right] \quad z}{[4096][2][8192]},$$

即:

$$\frac{\hat{y}_1 \quad y_0 \quad x \quad z}{[4096][2][8192][8192]}.$$

这样, y 维度和 x 维度得到交换,数据按照 y_1 维度分布在集群节点上, x 维度方向的数据存储在各个节点之内.

此时,各个节点可以分 $2(y_0$ 维度)次,将 8192×8192 的数据拷贝到 GPU 设备存储器上并进行正方形的数据转置,使得 x 维度的数据完全连续,得到:

$$\frac{\hat{y}_1 \quad y_0 \quad z \quad x}{[4096][2][8192][8192]}.$$

在使用 CUFFT 进行 x 维度的 1 维 FFT 计算之后,反方向重复整个数据移动过程,将数据恢复到初始的位置.

以上算法的 Parray 代码实现如图 14 所示:第 1 行声明自然数组类型 *HSTS*,代表整个数据在集群上的维度划分;第 2 行声明自然数组类型 *HST*,引用到 *HSTS_1* 维度,代表数据在一个节点上的维度划分;第 5 行~第 7 行,声明 MPI 进程数组类型 *MYTA* 以及混合数组类型 *S* 和 *T*,注意,*S* 和 *T* 中的对应维度进行了调整,之后,将使用其进行集群上数据的转置通信;进入 *FFT3D_FORWARD* 的执行代码,第 8 行~第 12 行,各个节点上分 2 次将 y 维度和 z 维度的 8192×8192 的数据传输到 GPU 上进行计算;第 13 行,copy 子程序利用之前声明的混合数组类型 *S* 和 *T* 进行算法中的集群之间的通信;第 14 行~第 17 行,各个节点将本地 x 维度的数据传输到 GPU 上进行 1 维 FFT 的计算;最后,反方向重复整个数据移动过程,将数据恢复到初始的位置.

```

1.  $parray {pinned float2[4096][[2][[4096][2][8192]]]} HSTS
2.  $parray {pinned float2[#HSTS_1]} HST
3.
4.  $subprog FFT3D_FORWARD(N,P,HST,DEV) {
5.    $parray {mpi[4096]} MYTA
6.    $parray {[[#MYTA][#HST_0][#HST_1_0][#HST_1_1][#HST_1_2]} S
7.    $parray {[[#HST_1_0][#HST_1_1][#MYTA][#HST_0][#HST_1_2]} T
8.    for (int slicei=0; slicei<2; slicei++) {
9.      $copy HST_1(host+$HST_0[slicei]) to DEV(dev)
10.     GPU_SAFE(cuffiExecC2C(planb.dev,dev,CUFFT_FORWARD),"FFT X-Y")
11.     $copy DEV(dev) to HST_1(buf+$HST_0[slicei])
12.    }
13.    $copy S(buf) to T(host)
14.    for (int slicei=0; slicei<2; slicei++) {
15.      $copy HST_1(host+$HST_0[slicei]) to DEVT(dev)
16.      $TRANPOSE_OUTP(dev,dev2,$DEV)
17.      GPU_SAFE(cuffiExecC2C(planc.dev2,dev2,CUFFT_FORWARD),"FFT Z")
18.      $TRANPOSE_OUTP(dev2,dev,$DEV)
19.      $copy DEV(dev) to HST_1(buf+$HST_0[slicei])
20.    }
21.    $copy S(buf) to T(host)
22.  }
23.  $end

```

Fig. 14 3D FFT cluster code in Parray

图 14 Parray 书写的集群 3 维 FFT 代码

可以看到:在图 14 中,GPU 集群 3 维 FFT 的核心代码不到 30 行,而且程序的表述也是十分自然和清晰的,

没有出现晦涩难懂的处理维度变化的嵌套循环的代码。

使用 Parray 实现的 GPU 集群 3 维 FFT(下称 PKUFFT)在天河 1A 上进行了性能测试,并与 Intel MKL 10.3.1.048 进行了比较.图 15 给出了它们在对不同规模的单精度复数数据做 3 维 FFT 时的性能,可以看出,PKUFFT 的性能远远超出 MKL;图 16 显示出:与 MKL 相比较,PKUFFT 具有更好的性能延展性.

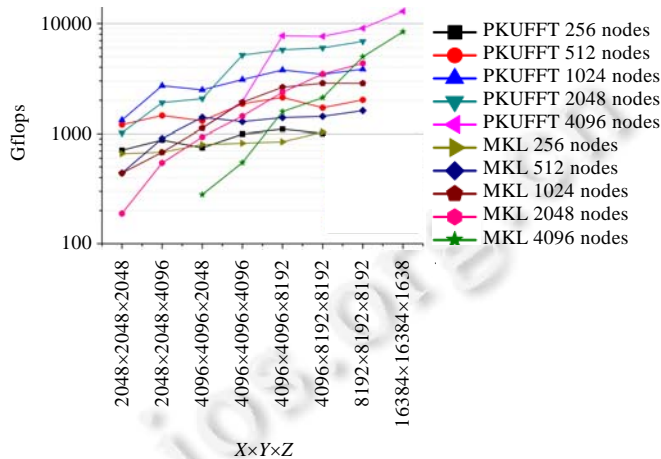


Fig.15 Performance comparison result

图 15 Parray 3 维 FFT 代码同系统性能比较

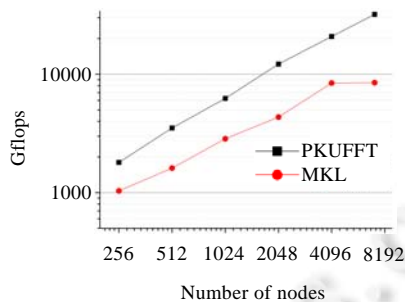


Fig.16 Speed-up ratio of Parray 3D FFT

图 16 Parray 3 维 FFT 代码加速比

5 结束语

本文介绍了数组维度类型程序设计方法,通过该方法,可以对异构集群科学计算问题中复杂的数组维度处理过程进行清晰的描述.通过使用 Parray 语言的新型数组类型,可以在编码阶段对数组维度处理过程进行简洁的表示.同时,本文还介绍了如何使用数组维度类型程序设计方法和 Parray 语言实现天河 1A 系统上的大规模 3 维 FFT,该算法得到了良好的性能和可延展性.

References:

- [1] TOP500. <http://www.top500.org/>
- [2] Cui X, Chen YF, Mei H. Improving performance of matrix multiplication and FFT on GPU. In: Proc. of the 15th Int'l Conf. on Parallel and Distributed Systems (ICPADS 2009). 2009. [doi: 10.1109/ICPADS.2009.8]
- [3] Cui X, Chen YF, Zhang CY, Mei H. Auto-Tuning dense matrix multiplication for GPGPU with cache. In: Proc. of the 16th Int'l Conf. on Parallel and Distributed Systems (ICPADS 2010). 2010. [doi: 10.1109/ICPADS.2010.64]

- [4] Chen YF, Cui X, Mei H. Large-Scale FFT on GPU clusters. In: Proc. of the 24th Int'l Conf. on Supercomputing (ICS 2010). 2010. [doi: 10.1145/1810085.1810128]
- [5] Govindaraju NK, Lloyd B, Dotsenko Y, Smith B, Manferdelli J. High performance discrete Fourier transforms on graphics processors. In: Proc. of the 2008 ACM/IEEE Conf. on Supercomputing (SC 2008). 2008. [doi: 10.1109/SC.2008.5213922]
- [6] Micikevicius P. 3D finite difference computation on GPUs using CUDA. In: Proc. of the 2nd Workshop on General Purpose Processing on Graphics Processing Units (GPGPU-2). 2009. [doi: 10.1145/1513895.1513905]
- [7] Ryoo S, Rodrigues CI, Bagnsorkhi SS, Stone SS, Kirk DB, Hwu WW. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In: Proc. of the 13th ACM SIGPLAN Symp. on Principles and practice of parallel programming (PPoPP 2008). 2008. [doi: 10.1145/1345206.1345220]
- [8] Volkov V, Kazian B. FFT prototype. <http://www.cs.berkeley.edu/volkov/>
- [9] Dotsenko Y, Bagnsorkhi SS, Lloyd B, Govindaraju NK. Auto-Tuning of fast Fourier transform on graphics processors. In: Proc. of the 16th ACM Symp. on Principles and Practice of Parallel Programming (PPoPP 2011). ACM Press, 2011. [doi: 10.1145/1941553.1941589]
- [10] Govindaraju NK, Lloyd B, Dotsenko Y, Smith B, Manferdelli J. High performance discrete Fourier transforms on graphics processors. In: Proc. of the 2008 ACM/IEEE Conf. on Supercomputing (SC 2008). 2008. [doi: 10.1109/SC.2008.5213922]
- [11] Nukada A, Matsuoka S. Auto-Tuning 3-D FFT library for CUDA GPUs. In: Proc. of the Conf. on High Performance Computing Networking, Storage and Analysis (SC 2009). 2009. [doi: 10.1145/1654059.1654090]
- [12] Nukada A, Ogata Y, Endo T, Matsuoka S. Bandwidth intensive 3-D FFT kernel for GPUs using CUDA. In: Proc. of the 2008 ACM/IEEE Conf. on Supercomputing (SC 2008). 2008. [doi: 10.1109/SC.2008.5213210]
- [13] MPFFT: An auto-tuning FFT library for OpenCL GPUs. [doi: 10.1007/s11390-013-1314-8]
- [14] del Mundo C, Feng WC. Towards a performance-portable FFT library for heterogeneous computing. In: Proc. of the 11th ACM Conf. on Computing Frontiers (CF 2014). 2014. [doi: 10.1145/2597917.2597943]
- [15] Fastest Fourier transform in the west. <http://www.fftw.org/>
- [16] Intel Corp. <http://software.intel.com/en-us/Intel-mkl/>
- [17] Parallel three-dimensional fast fourier transforms. <http://www.sdsc.edu/us/resources/p3dfft/>
- [18] Pekurovsky D. P3DFFT: A framework for parallel computations of Fourier transforms in three dimensions. *SIAM Journal on Scientific Computing*, 2012,34(4):C192–C209. [doi: 10.1137/11082748X]
- [19] Ayala O, Wang LP. Parallel implementation and scalability analysis of 3D fast Fourier transform using 2D domain decomposition. *Parallel Computing*, 2013,39(1):58–77. [doi: 10.1016/j.parco.2012.12.002]
- [20] Takahashi D. An implementation of parallel 3-D fft with 2-D decomposition on a massively parallel cluster of multi-core processors. In: Proc. of the Parallel Processing and Applied Mathematics. LNCS 6067, Berlin, Heidelberg: Springer-Verlag, 2010. [doi: 10.1007/978-3-642-14390-8_63]
- [21] Eleftheriou M, Fitch BG, Rayshubskiy A, Ward TJC, Germain RS. Scalable framework for 3d ffts on the blue gene/l supercomputer: Implementation and early performance measurements. *IBM Journal of Research and Development*, 2005,49(2):457. [doi: 10.1147/rd.492.0457]
- [22] Sabharwal Y, Garg SK, Garg R, Gunnels JA, Sahoo RK. Optimization of fast Fourier transforms on the Blue Gene/L supercomputer. In: Proc. of the 15th Int'l Conf. on High Performance Computing (HiPC 2008). 2008. [doi: 10.1007/978-3-540-89894-8_29]
- [23] 2decomp&fft. <http://www.2decomp.org/>
- [24] Kandalla K, Subramoni H, Tomko K, Pekurovsky D, Sur S, Panda DK. High-Performance and scalable nonblocking all-to-all with collective offload on infiniband clusters: A study with parallel 3D fft. *Computer Science*, 2011,26(3-4):237–246. [doi: 10.1007/s00450-011-0170-4]
- [25] Rahimian A, Lashuk I, Veerapaneni S, Chandramowlishwaran A, Malhotra D, Moon L, Sampath R, Shringarpure A, Vetter J, Vuduc R, Zorin D, Biros G. Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures. In: Proc. of the 2010 ACM/IEEE Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC 2010). 2010. [doi: 10.1109/SC.2010.42]

- [26] Ishiyama T, Nitadori K, Makino J. 4.45 pflops astrophysical n -body simulation on k computer: The gravitational trillion-body problem. In: Proc. of the 2012 ACM/IEEE Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC 2012). 2012. [doi: 10.1109/SC.2012.3]
- [27] Song S, Hollingsworth JK. Designing and auto-tuning parallel 3-D FFT for computation-communication overlap. In: Proc. of the 19th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP 2014). 2014. [doi: 10.1145/2555243.2555249]
- [28] Bell C, Bonachea D, Nishtala R, Yelick K. Optimizing bandwidth limited problems using one-sided communication and overlap. In: Proc. of the 20th Int'l Parallel & Distributed Processing Symp. (IPDPS 2006). 2006. [doi: 10.1109/IPDPS.2006.1639320]
- [29] Fang B, Deng Y, Martyna GJ. Performance of the 3D fft on the 6D network torus qcdoc parallel supercomputer. Computer Physics Communications, 2007,176(8):531–538. [doi: 10.1016/j.cpc.2006.12.006]
- [30] Doi J, Negishi Y. Overlapping methods of all-to-all communication and fft algorithms for torus-connected massively parallel supercomputers. In: Proc. of the 2010 ACM/IEEE Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC 2010). 2010. [doi: 10.1109/SC.2010.38]
- [31] Nukada A, Sato K, Matsuoka S. Scalable multi-GPU 3-D FFT for TSUBAME 2.0 supercomputer. In: Proc. of the Int'l Conf. on High Performance Computing, Networking, Storage and Analysis (SC 2012). 2012. [doi: 10.1109/SC.2012.100]
- [32] Czechowski K, Battaglino C, McClanahan C, Iyer K, Yeung PK, Vuduc R. On the communication complexity of 3D FFTs and its implications for exascale. In: Proc. of the 26th ACM Int'l Conf. on Supercomputing (ICS 2012). 2012. [doi: 10.1145/2304576.2304604]
- [33] HPC. <http://icl.cs.utk.edu/hpc/index.html>
- [34] Chen YF, Cui X, Mei H. PARRAY: A unifying array representation for heterogeneous parallelism. In: Proc. of the 17th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP 2012). 2012. [doi: 10.1145/2145816.2145838]



崔翔(1975—),男,河南开封人,博士,副教授,主要研究领域为 GPU 加速集群的程序设计方法.



陈一峰(1973—),男,博士,研究员,博士生导师,主要研究领域为众核集群程序语言,语言理论.



李晓雯(1984—),女,讲师,主要研究领域为程序设计方法.