CC$:一种面向分布式众核平台的并行编程语言

吴峻峰[1,2] 许跃生[1,2,3] 张永东[1,2] 江 颖[1,2] 叶纬材[1,2]
(中山大学数学与计算科学学院 广州 510275)[1] (中山大学广东省计算科学重点实验室 广州 510275)[2]
(雪城大学数学系 纽约州 13244)[3]

摘 要 CC$是一种并行编程语言,目的是解决分布式众核并行计算机的编程困难。CC$的编程模型以Multi-BSP模型为基础,将分布式众核并行计算机的硬件架构抽象为3层。数据按照存储的层次和共享范围分为5类,以便在不同层次上提供共享。CC$还提出一类虚拟指令来解决不同层次之间的数据交换,实现数据访问的逻辑化描述。并行程序按照3层Multi-BSP超步嵌套执行。CC$具有统一的编程风格、内建的多层公共地址空间、数据访问请求的表达式描述和数据传输编译优化4大特点。测试表明,CC$程序的运行效率高,易学易用,大幅地缩短了开发周期。

关键词 分布式众核,并行计算,编程语言,Multi-BSP,并行编程模型
中图法分类号 TP312 **文献标识码** A

CC$:A Parallel Programming Language for Distributed Many-core Platforms

WU Jun-feng[1,2] XU Yue-sheng[1,2,3] ZHANG Yong-dong[1,2] JIANG Ying[1,2] YE Wei-cai[1,2]
(School of Mathematics and Computational Science, Sun Yat-sen University, Guangzhou 510275, China)[1]
(Guangdong Province Key Laboratory of Computational Science, Sun Yat-sen University, Guangzhou 510275, China)[2]
(Department of Mathematics, Syracuse University, Syracuse NY 13244, USA)[3]

Abstract We presented a parallel programming language CC$. CC$ is developed to reduce the programming complexity on distributed many-core systems. The programming model of CC$ is based on Multi-BSP model, which abstracts distributed many-core systems into three levels. Data on CC$ machines are classified into 5 categories by data locality and shared regions, that is convenient for data access among different levels. CC$ also presents some virtual directives to describe data access logically. The programs on CC$ machines are executed with Multi-BSP super-steps. There are four key features of CC$: unified programming style for all levels, built-in multi-level common address spaces, description of data access requests with expressions, compile-time optimization for data transport. The experiments show that CC$ is easy and effective.

Keywords Distributed many-core, Parallel computation, Programming language, Mult-BSP, Parallel programming model

1 引言

分布式众核计算是当今高性能计算的主流形式之一。分布式并行计算机是用网络连接的多台计算机,这些计算机节点分摊从总计算任务分划出的子任务,并通过网络交换数据,实现总计算任务的"分而治之"。众核是指计算机处理器芯片上集成的大量计算核心。分布式众核并行计算机就是计算节点上配置众核的分布式并行计算机。这样的硬件平台具有多层次的硬件特性:互连网络的多层次、内存缓存的多层次、处理单元的多层次。要发挥多层硬件的性能,并行算法必须是多层次的。

分布式众核多层架构的出现使并行编程的困难加剧,这些困难主要体现在3方面:1)学习过程长;2)过于底层化,与硬件高度相关,调试优化困难;3)难以在逻辑层面自然表达程序并发性的同时获得高运行性能。而面向分布式众核架构的并行程序设计至少有3个新问题:1)必须统一考虑节点内对应于众核特性的细粒度并行和节点间对应于网络特性的粗粒度并行;2)必须同时处理节点间、节点内多个处理器芯片间、芯片内多个核间等多个层次计算的数据调配和任务调度;3)要在上述的不同硬件层次上处理不同的性能瓶颈问题。

在众核、分布式和分布式众核架构上,传统的并行编程解决方案是多样化的。常见的并行编程解决方案有3类:1)串行编程语言+运行库,例如MPI[18]、PVM[8]、BSPonMPI[4]、BSPLib[10]、GlobalArrays[12];2)串行编程语言+编译指导性

到稿日期:2012-10-19 返修日期:2012-12-15 本文受广东省引进创新科研团队一计算科学科研团队(粤组函[2010]189号),中山大学广东省计算科学重点实验室(粤财教[2010]311号)资助。

吴峻峰(1980—),男,博士,讲师,主要研究方向为高性能计算、并行计算;许跃生(1957—),男,博士,教授,主要研究方向为小波分析、数值逼近、积分方程数值解、微分方程数值解、高性能计算、高精度图像处理,E-mail:xuyuesh@mail.sysu.edu.cn(通信作者);张永东(1962—),男,博士,副教授,主要研究方向为并行计算、计算机真实感图形技术、智能调度算法研究;江 颖(1980—),男,博士,讲师,主要研究方向为高维高精度傅立叶变换、高性能计算;叶纬材(1980—),男,博士,主要研究方向为高性能计算、并行计算、生物信息。

注释，例如 OpenMP[15]、OpenACC[16]；3）并行编程语言，例如 X10[11]、Chapel[7]、UPC[19]、CoArrayFortran[5]、CUDA[14]、OpenCL[13]、Cilk[1]、BEC[9,21]和 PPM[3]。

将这些解决方案用于分布式众核架构时，必须使用多种并行编程模型才能匹配多层硬件架构：不同层次的硬件使用不同的编程模型描述。并行编程模型是把逻辑层面的并行算法转化为底层硬件的并行计算程序的工具[20]。传统的并行编程模型如 MPI、OpenMP 等只能实现单层次的并行程序，新兴的并行编程模型如 CUDA、OpenCL 和 Cilk 也只关心众核芯片内部的并行程序，这导致了分布式众核计算软件的研发人员需要同时学习多个不同风格的并行编程模型，并混合使用相应的软件工具。这些复杂的并行编程解决方案导致了分布式众核并行编程的困难。

CC$ 编程语言的主要目的是解决分布式众核并行计算的软件研发困难。为方便并行算法的实现，CC$ 从以下几方面入手：引入多层次并行计算模型，为并行算法提供适应分布式众核硬件的表达方式；引入多层次逻辑共享数组，实现数据存储的虚拟化、层次化和访问统一化；引入多层次数据访问映射，实现方便、灵活、高效的数据交换机制，只需在逻辑上定义交换操作，无需在程序里编写数据交换细节，让编译器根据具体的交换请求生成高效的目标代码实现数据交换；简化 OpenCL 调用机制，无需使用与硬件相关的 OpenCL 驱动 API，就能直接调用 OpenCL 核函数。

本文第 2 节讨论 CC$ 语言的编程模型；第 3 节描述 CC$ 的语言特性；第 4 节给出 CC$ 编译优化机制的简介并与 BSP on MPI 进行性能对比；第 5 节通过一个综合实验说明 CC$ 的优良性能；最后总结全文。

2 分布式众核并行编程模型

与并行编程模型相关的 3 部分是硬件模型、存储模型和计算模型。硬件模型抽象地描述了编程语言所面向的目标硬件平台的基本结构与特征；存储模型在逻辑上描述了编程语言的数据分布及交换的方式；计算模型描述了程序在目标硬件平台上的并发执行过程。

CC$ 的编程模型是以 Leslie Valiant 的 Multi-BSP 模型[20]为基础的。Multi-BSP 模型用于刻画多层硬件架构上的并行算法。Multi-BSP 模型将一个并行计算硬件架构按照层次划分，每一层硬件都有 4 个属性：子计算单元数目、内存或 cache 大小、网络带宽和同步代价。Multi-BSP 模型比 BSP 模型[22]增加了硬件层次数，并在每层硬件加上了内存或 cache 大小的属性。这样的模型原是为多核架构设计的，同样也适用于分布式众核架构。Hou 等提出的 BSGP[23]将 BSP 模型应用于单节点的单 GPU 众核架构上，是 Multi-BSP 模型的一个特例。Savadi 提出的 Multi-DaC 模型[24]是 Multi-BSP 模型在分治算法上的特例。对于分布式众核架构，并行计算硬件至少包括 3 个层次：计算网络层、节点层和处理器芯片层。

Multi-BSP 模型虽然能较好地描述分布式众核架构上的并行计算，但是到目前还没有对应的软件解决方案。本文的 CC$ 语言在 Multi-BSP 的基础上，实现了一种适用于分布式众核架构的软件解决方案，不但可使算法设计和软件开发摆脱繁琐的硬件细节，而且能抓住充分发挥硬件性能的关键要素。

在硬件模型方面，CC$ 把硬件抽象为 3 个层次，由小到大分别是设备层、局部层和全局层，对应的处理单元分别称为设备单元、局部单元和全局单元。（1）设备层是对应芯片电路联结的所有核的总体[12]，每个设备单元常常是众核芯片。设备单元的任务按照设备线程的阵列分划并调度给各个核执行。（2）局部层是由共享内存联结的所有设备单元的总体，每个局部单元通常是计算节点。局部单元的任务按照设备单元的阵列分划并调度给各设备单元执行。（3）全局层是所有参与计算的局部单元的总体。全局任务按照局部单元的阵列分划并调度给局部单元执行。对于 GPU 集群来说，每个 GPU 芯片都是一个设备单元，每个计算节点是一个局部单元。

存储模型按照硬件模型相应分为 3 层：设备存储器、局部存储器和全局存储器。设备存储器在设备单元内，供设备单元中的所有核共享。局部存储器位于局部单元内，供局部单元中的所有设备单元访问。全局存储器由所有局部单元访问。以 GPU 集群为例，设备单元是 GPU，设备存储器是 GPU 显存。局部存储器对应的是每个计算节点的主内存。全局存储器是虚拟的，其数据实际分布在局部存储器中。用网络传输实现对全局数据的访问。

CC$ 把数据按照存储的层次和共享范围分为 5 种类别，按照共享程度由小到大排列，分别为设备私有数据、设备共享数据、局部私有数据、局部共享数据和全局数据。首先，设备存储器中的数据有一部分是只供所属的设备单元使用的，将这部分数据归为设备私有数据；还有一部分是要在不同的设备单元之间共享的，将这部分数据归为设备共享数据。共享过程中，有些系统（如 GPU 系统）可以借助于局部存储器来支持设备单元间的数据交换。类似地，局部存储器中的数据也分两类，分别是局部私有数据和局部共享数据，其中局部私有数据只供所属的局部单元使用，局部共享数据在不同的局部单元之间共享。最后，全局数据存储在虚拟的全局存储器中。在设备和局部层面区分私有和共享的数据类别，有助于私有数据的访问和维护节省共享开销，并使共享数据的访问能够得到编译时和运行时的双重优化。

在 CC$ 中，局部存储器为设备共享数据提供公共地址空间，全局存储器为局部共享数据提供公共地址空间，以方便这两层数据访问的算法描述。设备共享数据地址与局部公共地址空间之间的映射，以及局部共享数据地址与全局公共地址空间之间的映射，都由导入和导出语句定义。导入、导出语句用表达式描述映射，从而描述了驱动数据访问的数据访问请求关系，使 CC$ 的编译器能够根据数据访问请求关系和硬件特性灵活地生成高效率数据传输的目标代码。

在计算模型方面，CC$ 在 3 个硬件层次上的并行计算都由 BSP 超步[21]组成。（1）全局超步由局部计算和全局数据交换组成，全局数据交换导由入和导出语句描述。（2）局部超步由设备计算和局部数据交换组成，局部数据交换也由导入和导出语句描述。（3）设备超步由设备计算和设备层的同步组成。设备计算是设备单元内的多线程并行计算。

3 CC$ 语言特性

CC$ 并行编程语言是对 OpenCL、CUDA 等众核并行计

算编程语言进行 Multi-BSP 形式的分布式计算扩展的编程语言，能快速开发 C++中可以直接调用的并行计算模块，能够方便地与传统并行库(特别是 MPI)共存。使用 CC$ 与直接使用 MPI+OpenMP+OpenCL/CUDA 的传统方案相比，第一个优点是统一的编程风格。CC$ 在硬件模型的 3 个层次使用相同风格的编程模型，有利于算法研究；而传统方案在不同层次的编程模型差异很大，对算法研究造成了困难。第二个优点是内建的多层公共地址空间。CC$ 的多层公共地址空间方便算法描述和程序开发；传统方案没有内建多层公共地址空间，需要在程序中显式指定各层数据来源和去处，使算法和程序复杂化，自动优化的空间小。为了统一数据访问请求的描述和方便优化，CC$ 引入了数据访问请求的表达式描述，这是第三个优点。CC$ 的数据交换是用表达式描述交换请求的，而不是像传统方案那样显式指定交换数据的一系列通信操作，这种表达式描述大幅降低了编程的复杂性，并为数据交换的编译优化提供很大的空间，配合 CC$ 的编译优化技术，能为 BSP 数据交换带来的明显优化效果。CC$ 的数据传输编译优化是第四个优点。

在编程风格方面，CC$ 并行编程语言在硬件模型的 3 个硬件抽象层次，都使用核函数(kernel function)的方式来组织计算任务，并使用 BSP 风格表达任务并行。所谓核函数，是指在给定阵列的处理单元上共同执行的一种特殊的函数。核函数除了通常的函数参数，还必须有执行核函数的处理单元在阵列中的坐标参数。根据坐标参数的不同，核函数执行从该层次的总任务分划出来的不同的子任务。核函数的概念来自于 CUDA 和 OpenCL 的内核语言。

在公共地址空间方面，CC$ 允许通过导入、导出语句，从局部数组导入数据到设备共享数组，把设备共享数组的数据导出到局部数组，从全局数组导入数据到局部共享数组，把局部共享数组的数据导出到全局数组。不管是远程数据还是本地数据，都可以在数据导入、导出语句中直接用元素标号访问。因此，局部数组的元素标号为设备共享数据提供了公共地址空间，全局数组的元素标号为局部共享数据提供了公共地址空间。

在数据访问请求描述方面，CC$ 在导入、导出语句中支持用集合表达式描述数据映射关系。集合表达式是包含集合变量的表达式。在 CC$ 中，集合变量是变量名前带“$”符号的变量，并且在支持集合表达式的语句中定义。例如在下面的导入语句中，localA 是局部共享数组，globalA 是全局数组，$i 和 $j 是这个语句定义的两个集合变量，它们的取值范围分别是 localA 第 0 维、第 1 维的元素标号取值范围，在等号右端给出的是每个 localA[$i, $j]所对应的 globalA 元素。

import localA[$i, $j]=globalA[$(s_0+t_0*\$i)*\$j$, $(s_1+t_1*\$j)/(\$i+1)$]

定义集合变量时，也可以指定其它的取值范围。例如，下面的导入语句指定了 $i 的取值范围从 3 到 m(包含 3 和 m)。

import localA[$i=3:m, $j]=globalA[$i * $j, $i + $j]

其中，s_0、t_0、s_1、t_1 都是局部变量。

有关 CC$ 的数据传输编译优化的内容将在下一节详细介绍。

4 CC$ 的数据传输编译优化机制

在 MPI+OpenMP+CUDA/OpenCL 的传统方案中，无论是基于 MPI 还是 BSP，全局层面和局部层面的数据传输代码都主要是手工编写的，显式指定了传输操作。这些方案虽然编程灵活，但编程复杂，而且性能优化的负担几乎都由程序员承担。这个问题在全局层面尤其明显。全局层面的数据传输不但需要发送和接收数据，还涉及到数据同步、数据打包和数据拆包等方面。传统方案的程序显式规定了所有细节，没有留下调整的空间，使得编译器无法对数据传输进行有效的自动优化。

在 CC$ 中，数据传输是数据访问请求驱动的。各层处理单元只需列出计算涉及的数据，无需指定这些数据的传输过程细节，传输过程的具体操作由 CC$ 编译器在编译时代为决定。这种描述降低了编程的复杂性，给了 CC$ 编译器很大的编译优化空间，大幅度减轻了程序员的负担。下面针对全局层面的数据传输，介绍 CC$ 的编译优化机制。

首先，CC$ 把数据分成设备私有、设备共享、局部私有、局部共享、全局这 5 类，使得编译器可以有选择地为共享和全局数据生成副本维护的代码。这里的副本，是指异地数据在本地的一个拷贝。这些副本可以避免很多常用数据的反复传输。为了确保副本维护的开销不影响并行计算的效率，编译器在选择是否生成副本时综合地考虑了数据的规模和类别。

第二，在上述数据副本的帮助下，CC$ 可以高效率地在本地进行异地表达式的计算，使得导入、导出语句的请求表达式在本地、异地求值几乎没有效率上的差别。

第三，CC$ 编译器可以根据请求(集合)表达式的具体结构，制定收集访问请求的算法方案，并针对性地进行优化。请求表达式是集合表达式，由导入、导出语句定义。这个算法方案使用了下面 3 种技术。1)编译器生成代码，根据请求表达式，帮助本地节点尽可能在无需通讯的情况下，得出涉及的集合变量哪些取值与该节点相关，哪些取值与之无关，这样本地节点就可以尽可能地只求相关的值。2)编译器能找出集合表达式中的公共部分，这些公共部分对于集合变量的不同取值保持不变，因此可以事先计算，反复使用。3)对于多维数组的数据访问请求，每一维有一个集合表达式，编译器能够判断维与维之间的关系，适当地把请求求值过程降维处理。例如当各维表达式使用互不相同的集合变量时，各维的求值可以分开进行，然后把结果合并。这类似于函数张量积的快速数值计算过程。

第四，这种基于表达式的快速收集访问请求过程需要的数据通信量极小。在很多情况下，由于数据副本完整，收集访问请求过程无需通信。这让 CC$ 略去了传统 BSP 支持库中为获取异地数据而向异地节点发送远程内存访问申请的过程，使得 CC$ 的数据传输过程节省了大量的网络带宽，因而传输效率得到明显的提高。

最后，这种基于表达式的快速收集访问请求过程给了数据同步、打包、拆包和收发非常大的灵活性，使得 CC$ 编译器可以根据硬件特性针对性地制定不同的传输操作流程。例如，中山大学“南方一号”集群的特点是点对点通信比较高效，多对多通信不管是 all to all 方式、非阻塞方式还是阻塞方式，

都比较低效。因此，在“南方一号”集群上，CC$会尽量地用多个点对点通信代替多对多通信。

为了展示CC$对数据传输的编译优化效果，在“南方一号”上我们选择了BSP on MPI库[4]作一系列传输效率的对比。在Infiniband通用计算网络上，BSP on MPI相对于其它传统的BSP支持库有较高的传输效率。

表1列出了在两个三维全局数组A、B之间的数据传输需要的时间。这两个数组三维的长度都是512。计算节点按照$q\times q$的阵列排列。A、B的第0维沿计算节点阵列的第0维平均分划，第1维沿计算节点阵列的第1维平均分划，第2维不分划。表中的行对应测试的项目，列对应数据传输时间。表中CC$和BSP分别表示CC$程序和BSP on MPI v0.3程序使用4、9和16节点时的数据交换时间。数据交换时间统计了包括数据准备、收发和相关计算的全部过程的用时。Rate表示比率：BSP/CC$。数据传输时间越短，表明数据传输效率越高。

表1 CC$与BSP on MPI的数据交换时间比较（单位：秒）

测试项目	4个节点			9个节点			16个节点		
	CC$	BSP	Rate	CC$	BSP	Rate	CC$	BSP	Rate
数组拷贝	0.06	5.344	89.1	0.0438	2.026	46.3	0.027	1.287	47.7
数组转置021	0.341	7.184	21.1	0.318	3.408	10.7	0.126	2.058	16.3
数组转置102	0.547	4.867	8.9	0.299	2.158	7.2	0.143	2.058	14.4
数组转置120	0.835	5.771	6.9	0.514	3.437	6.7	0.301	2.022	6.7
数组转置201	0.805	5.784	7.2	0.783	3.741	4.8	0.414	2.459	5.9
数组转置210	0.416	5.797	13.9	0.404	3.826	9.5	0.149	2.368	15.9
数组翻转100	0.55	7.249	13.2	0.321	3.18	9.9	0.142	1.738	12.2
数组翻转010	0.547	7.452	13.6	0.329	3.13	9.5	0.142	1.766	12.4
数组翻转001	0.061	6.914	113.3	0.041	3.038	74.1	0.252	1.429	5.7
数组翻转110	0.551	7.25	13.2	0.752	3.128	4.2	0.143	1.633	11.4
数组翻转011	0.562	7.469	13.3	0.337	2.731	8.1	0.146	1.751	12.0
数组翻转101	0.564	7.256	12.9	0.397	3.162	8.0	0.147	1.739	11.8
数组翻转111	0.561	7.179	12.8	0.769	3.093	4.0	0.143	1.682	11.8
数组随机读写100	0.274	7.231	26.4	0.285	3.13	11.0	0.116	1.544	13.3
数组随机读写010	0.262	7.219	27.6	0.263	3.163	12.0	0.111	1.578	14.2
数组随机读写001	0.06	6.968	116.1	0.04	3.028	75.7	0.025	1.328	53.1
数组随机读写110	0.372	7.337	19.7	0.447	2.085	4.7	0.14	0.986	7.0
数组随机读写011	0.247	7.349	29.8	0.252	3.179	12.6	0.108	1.596	14.8
数组随机读写101	0.248	7.352	29.6	0.255	3.168	12.4	0.108	1.588	14.7
数组随机读写111	0.359	7.388	20.6	0.428	2.588	6.0	0.137	1.028	7.5

表中的“数组转置p_0 p_1 p_2”是指A的第p_0、p_1、p_2维分别转到B的第0、1、2维。数据翻转就是将数据按照某一维度的中心点做镜像交换，如长度为512的数组，第i号元素就与$511-i$号交换。表中的“数组翻转b_0 b_1 b_2”是指，当b_i为0时第i维不翻转，为1时翻转，$i=1,2,3$。表中“数组随机访问b_0 b_1 b_2”是指，当b_i为0时第i维不随机访问，使用B的元素的第i维标号作为A的元素的第i维标号，当b_i为1时表示第i维标号是一个随机数，$i=1,2,3$。此外，不同维标号的随机数互不相关。

表1中的数组拷贝的CC$程序的运行速度是BSP on MPI的46倍以上；数组转置的CC$程序的速度是BSP on MPI的4.8倍以上，最高达21倍；数组翻转的CC$程序的速度是BSP on MPI的5.7倍以上，最高达113倍；数组随机访问的CC$程序的速度是BSP on MPI的4.7倍以上，最高达116倍。数组拷贝和数组随机读写001等只访问本地数据的项目，CC$的编译优化完全避免了通信，因此效率很高；在随机访问的测试项目中，CC$能够利用数据访问请求在数组各维的统计独立性，极大地降低了数据访问请求过程中所需的通信量，节省了远程内存访问申请所占用的大量网络带宽，因此有明显的传输效率优势。

综上所述，CC$的数据传输编译优化机制灵活，可根据不同情况进行优化，而BSP on MPI缺少编译优化的环节。因此CC$在传输效率上比BSP on MPI有较明显的优势。下面是测试所用的CC$主要代码示例，其中localB是数组B的局部部分，s_0、s_1分别是localB第0、1维开始位置在数组B中的序号，N是数组单维的长度。

数组转置021的数据访问请求表达式代码：

import localB[$i,$j,$k]=A[$s_0+$i,$k,s_1+j]

数组翻转011的数据访问请求表达式代码：

import localB[$i,$j,$k]=A[$s_0+$i,$N-1-s_1-$j,$N-1-$k]

数组随机访问101的数据访问请求表达式代码：

import localB[$i,$j,$k]=A[(local shared)R0[$i],s_1+j,(local shared)R2[$k]]

上面R0和R2是两个局部私有数组，使用(local shared)操作符进行强制共享域转换，转成局部共享数组，让异地节点可以使用R0、R1中的数据计算获得localB对A的请求。

测试项目中的CC$访问请求表达式都很简短，而BSP on MPI的程序的相应代码要39行。除此之外，CC$的编译优化机制对很复杂的访问请求表达式依然能够适用。

5 综合实例测试

为验证CC$的易用性，本文选择了具有代表性的BPX多重网格预条件算子[2]来加速GMRES(广义最小残差法)[16]求解器，求解高阶有限体积法[5]生成的稀疏线性方程组。在每个GMRES迭代中，使用8个正交化的单位向量来搜索方程迭代解的收敛方向。

本文比较了该实例使用C++开发串行程序与使用CC$开发并行程序的时间。串行程序花费了两周实现，CC$实现的并行化版本只花费了一周时间。由于该实例有相当多的数据通信和任务调度，并行化该实例的研发效率达到了预期的效果。

该实例程序包含两部分源文件：CC$文件和C++文件。编译时，CC$文件经过CC$源到源编译器转换成C++模块，供C++文件调用。OPENCL核程序代码作为设备层的核函数包含在CC$文件中。

实验运行在“南方一号”集群上。该集群每个节点上有2个六核的Intel Xeon E5620 CPU(工作频率2.6GHz)、6个NVIDIA C2050 GPU，以及72GB的主内存。计算网络是40Gbs的Infiniband。

测试结果如图1、图2所示。由于实例所用的数据量太大，必须使用至少4个节点来进行计算(图中与串行相比的并行加速比是根据大小缩小16倍的数据用串行程序算所消耗的时间推算出来的)。使用了两组测试数据，一组是包含超过

10亿个未知量的方程，另一组是包含超过16亿个未知量的方程。

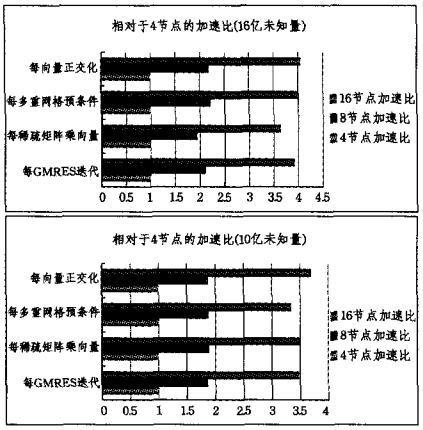

图1

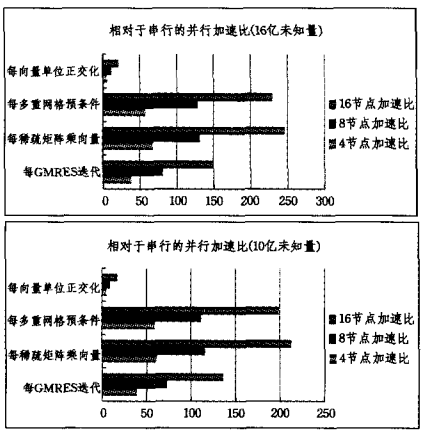

图2

测试过程中，该并行GMRES求解器对16亿未知量的方程用10个迭代收敛到6.3E－9的精度，对10亿未知量的方程用9个迭代收敛到2.9E－9的精度，收敛所需的迭代数和精度都与使用多少个节点进行并行计算无关。附着节点数的增加，该并行GMRES求解器获得接近线性的并行加速比。

结束语 本文提出的CC$扩展了CUDA和OpenCL，是易学易用的并行编程语言，适用于分布式众核计算机。使用CC$的优点在于：1)统一的编程风格：CC$在硬件模型的三个层次使用相同风格的编程模型，有利于算法研究；2)内建的多层公共地址空间：CC$ 的多层公共地址空间方便算法描述和程序开发；3)数据访问请求的表达式描述：不但统一数据访问请求的描述，而且方便优化；4)CC$的数据传输编译优化：能为数据交换带来明显的优化效果。我们期望通过CC$研出发多种适合超级计算机的应用软件。

注：CC$并行编程语言的第一版已经发布在中山大学的南方一号集群上。

参考文献

[1] Blumofe R, Joerg C, Kuszmaul B, et al. Cilk: An efficient multithreaded runtime system[C]// Proceedings of the fifth ACM SIGPLAN symposium on Principles and practice of parallel programming. 1995:207-216

[2] Bramble J, Pasciak J, Xu J. Parallel multilevel preconditioners [J]. Mathematics of Computation, 1990, 55(191): 1-22

[3] Brightwell R, Heroux M, Wen Z, et al. Parallel phase model: a programming model for high-end parallel machines with many-cores [C] // International Conference on Parallel Processing. 2009:92-99

[4] BSP on MPI home page [OL]. http://bsponmpi.sourceforge.net/

[5] C-A F W G. Co-Array FORTRAN home page [OL]. www.co-array.org

[6] Chen Z, Wu J, Xu Y. High order finite volume methods for elliptic boundary value problems [J]. Advances in Computational Mathematics, 2012, 37(2): 191-253

[7] Cray. Chapel — the cascade high-productivity language [OL]. http://chapel.cs.washington.edu/

[8] Geist A, et al. PVM home page [OL]. www.csm.ornl.gov/pvm/pvm home.html

[9] Heroux M, Wen Z, Wu J. Initial experiences with the BEC parallel programming environment[C]// The 7th International Symposium on Parallel and Distributed Computing. 2008:205-212

[10] Hill J. The Oxford BSP toolset [OL]. www.bspworldwide.org/implmnts/oxtool/

[11] IBM. The X10 Programming Language [OL]. http://x10-lang.org/

[12] Neplocha J, Harrison R J, Littlefield R J. Global arrays: a non uni-form memory access programming model for high performance computers[J]. Journal of Supercomputing, 1996, 10(2): 197-220

[13] Khronos OpenCL Working Group. The OpenCL specification [OL]. 2008

[14] NVIDIA. CUDA programming guide [OL]. 2008

[15] O A R B. OpenMP Fortran application interface version 1.1 [OL]. www.openmp.org

[16] OpenACC. The OpenACC standard [OL]. http://www.openacc-standard.org, 2012

[17] Saad Y, Schultz M. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems[J]. SIAM Journal on Scientific and Statistical Computing, 1986, 7(3): 856-869

[18] Snir M, Otto S, Huss-Lederman S, et al. MPI-the complete reference, volume 1, the MPI core[M]. The MIT Press, 1998

[19] Consortium U. UPC language specification (v1.2) [OL]. http://www.gwu.edu/upc/documentation.html

[20] Valiant L. A bridging model for multi-core computing[J]. Journal of Computer and System Sciences, 2011, 33(8): 154-166

[21] Wen Z, Wu J, Xu Y. BEC specification and programming reference[R]. SAND2007-7617. Albuquerque, NM USA: Sandia National Laboratories, 2007

[22] Valiant L G. A Bridging Model for Parallel Computation[J]. Communications of the ACM, 1990, 33(8): 103-111

[23] Hou Q, Zhou K, Guo B. BSGP: Bulk-Synchronous GPU programming[J]. ACM Transactions on Graphics, 2008, 27(3): 1-13

[24] Savadi A, Moradi M, Deldari H. Multi-DaC programming model: a variant of multi-BSP model for divide-and-conquer algorithms [C]// Proceedings of the 7th Workshop on Declarative Aspects and Applications of Multicore Programming (DAMP'12). ACM, New York, NY, USA, 2012:41-46