

并行原型系统上 BFS 算法设计实现与测试分析^{*}

衡冬冬,唐玉华,易晓东,刘向阳,周 侗

(国防科学技术大学计算机学院,湖南 长沙 410073)

摘 要:相对于传统应用,大数据应用表现出并行性高、访存数据量大、访存模式不规则、程序访存时空局部性差等特性,对传统的计算机体系结构提出了新的挑战。Graph500 是评测计算机系统大数据处理能力的基准测试排名,BFS 算法是 Graph500 的核心程序,是典型的数据密集型应用。从 1-D 数据划分、优化的混合算法设计和远程通信方式设计三个方面开展研究,在课题组设计的大数据处理并行结构原型系统上设计实现了多节点的并行 BFS 算法,在 2^{22} 顶点、 2^{26} 边的数据规模下取得了 803.8 MTEPS 的性能,并在此基础上进行多节点并行 BFS 算法的性能测试分析,为进一步的研究工作奠定了基础。

关键词:大数据处理;Graph500;并行 BFS;并行结构原型系统;性能测试分析

中图分类号:TP302

文献标志码:A

doi:10.3969/j.issn.1007-130X.2017.01.003

Implementation and performance analysis of BFS algorithm on a parallel prototype system

HENG Dong-dong, TANG Yu-hua, YI Xiao-dong, LIU Xiang-yang, ZHOU Tong

(College of Computer, National University of Defense Technology, Changsha 410073, China)

Abstract: Compared with traditional applications, big data applications present have diffident characteristics, such as high parallelism, large volumes of data, irregular memory access mode, and poor temporal locality, which bring new challenges to traditional computer architectures. Graph500 which focuses on data intensive loads, is a rating of supercomputer systems. And the Bread First Search (BFS) algorithm is the core of the benchmark. Based on the parallel prototype system designed for big data applications, we implement the BFS algorithm from the perspectives of 1-D partitioning, optimized hybrid algorithm and convenient remote communication design. The performance achieves 803.8 MTEPS under the scale of an input graph of 2^{22} vertexes and 2^{26} edges. Then we analyze the experiment performance and provide a reference for future wok.

Key words: big data processing; Graph500; parallel BFS; parallel prototype system; performance analysis

1 引言

随着大数据时代的到来,数据正在以前所未有的速度增长。快速分析与处理大规模数据、挖掘海量数据背后的潜藏价值成为人们关注的重点。在此背景下,图分析处理成为近年来研究的热点,互

联网、社交网络、生物系统等领域的很多问题都可以抽象成图问题来分析解决。2010 年 6 月,美国圣地亚国家实验室联合几大 IT 公司定义并发布了一个新的基准测试排名 Graph500^[1],此基准测试以 BFS(Breadth First Search)算法为核心程序来评测计算机系统处理大数据应用的能力。

随着数据规模快速增长,互联网数据分析、商

^{*} 收稿日期:2016-08-20;修回日期:2016-10-20

基金项目:高性能计算国家重点实验室开放课题(201302-01,201513-01)

通信地址:410073 湖南省长沙市国防科学技术大学计算机学院

Address: College of Computer, National University of Defense Technology, Changsha 410073, Hunan, P. R. China

业智能分析、国家经济运行分析、科学研究、国家空天一体的安全分析等许多重要领域对大量数据处理和分析的高性能和时效性需求愈来愈高。而传统的计算机系统结构难以满足大数据处理的这种需求,因此分析大数据应用的负载特点,并设计适用于大数据处理的新型计算机体系结构有着重要的意义。

本文主要分为五个部分,第1节介绍 Graph500 基准测试程序和 BFS 算法;第2节介绍可扩展的大数据处理并行结构原型系统;第3节从 1-D 数据划分设计、优化的混合算法设计和远程通信方式设计三个方面介绍并行结构原型系统上 BFS 算法的设计实现;第4节介绍在并行结构原型系统上 BFS 算法的性能测试与分析;第5节对研究工作进行总结。

2 Graph500 基准测试程序和 BFS 算法

为指导大数据处理硬件体系结构与软件支撑系统的设计,在 2010 年的 SC2010 会议上,美国圣地亚国家实验室与 Intel、IBM、AMD、NVIDIA 和 Oracle 合作定义并发布了一个新的基准测试排名 Graph500。Graph500 基准测试以 BFS 算法为核心程序,利用图论分析超级计算机在模拟生物、安全、社会以及类似复杂问题时的吞吐量,并进行排名。

2.1 Graph500 基准测试程序

如图 1 所示,Graph500 基准测试程序由图生成、图建立、BFS 搜索与验证、结果输出四部分组成。

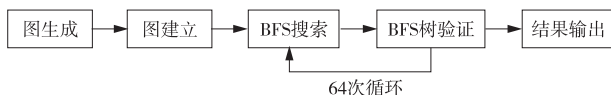


Figure 1 Flow diagram of the Graph500 benchmark

图 1 Graph500 基准测试程序流程示意图

(1)图生成:程序通过 Kronecker 图生成器生成一系列边元组信息,每一条边由一个顶点组合 $\langle StartVertex, EndVertex \rangle$ 表示, $StartVertex$ 、 $EndVertex$ 分别表示边的两个端点。图的规模是由用户输入的参数 $SCALE$ 、 $edgefactor$ 确定的,其中, $SCALE$ 指示图的顶点规模, $edgefactor$ 指示每个顶点连接边的平均数量, $N = 2^{SCALE}$ 表示输入图的顶点数目, $M = edgefactor * N$ 表示输入图的边数目。

(2)图建立:此过程将之前生成的边元组信息

转化成任意表示图的数据结构,生成的数据结构不能被后续部分改变。此过程的执行时间称为图的建立时间,不计入性能计算中。不同数据结构的选择将影响后续的 BFS 算法搜索效率及程序的执行性能。

(3)BFS 搜索与验证:程序随机生成一个根顶点,并以此为源点对整个图进行 BFS 搜索,记录每个顶点的前驱顶点作为搜索结果,并验证搜索得到的 BFS 生成树是否与原图信息匹配。如图 1 所示,根顶点生成、BFS 搜索和 BFS 生成树验证过程将循环 64 次。程序对 BFS 搜索部分计时,作为程序执行性能的衡量指标。BFS 算法的选择将影响程序的执行性能。

(4)结果输出:Graph500 用每秒遍历的边数 TEPS(Traversed Edges Per Second)来衡量程序的执行性能,由生成图边数 M 除以 BFS 搜索部分的时间 T 得到。

根据 Graph500 基准测试程序输入参数的不同,可以将求解问题划分为六个不同的数据规模^[1],分别为 Toy、Mini、Small、Medium、Large 和 Huge。

2.2 BFS 算法

2.2.1 串行 BFS 算法

BFS 算法是 Graph500 基准测试的核心程序。给定图 $G\langle V, E \rangle$ 和根顶点 $root$, BFS 可以搜索到由 $root$ 出发能够到达的所有顶点,且与 $root$ 距离为 k 的顶点一定先于距离为 $k+1$ 的顶点被搜索到。算法 1 是串行的 top-down BFS 算法,算法从 $root$ 开始搜索,得到 $parent$ 数组($parent$ 表示 BFS 生成树信息,保存顶点的前驱顶点,其中根顶点 $root$ 的 $parent$ 为其自身)。变量 F (Frontier) 表示当前待搜索的顶点集合, NF (Next Frontier) 表示下一层待搜索的顶点集合, $visit$ 表示每个顶点的访问信息。

算法初始时将 $root$ 加入 F , 然后遍历所有 $frontier$ 的邻居顶点,更新其中未访问顶点的 $visit$ 、 $parent$ 信息并将该顶点加入 NF 集合,每层结束时将 NF 赋给 F 并将 F 清空,循环执行直到 F 为空。

算法 1 串行 top-down BFS 算法

Input: $G = \langle V, E \rangle$ // the graph
 $root$: // root vertex
 $Var: F$: // frontier
 NF : // next frontier
 $Visit$: // Vertex is visited

Output: *parent*; // parent map

```

1   $F \leftarrow \emptyset; NF \leftarrow \emptyset;$ 
2  for each  $v$  in  $V$ 
3     $visit[v] = 0; parent[v] = -1;$ 
4     $visit[root] = 1; F \leftarrow F \cup \{root\};$ 
5     $parent[root] = root;$ 
6    while  $F! = \emptyset$ 
7      for each  $v$  in  $F$ 
8        for  $w \in E(v)$ 
9          if  $visit[w] = 0$ 
10              $visit[w] = 1;$ 
11              $parent[w] = v;$ 
12              $NF \leftarrow NF \cup \{w\};$ 
13      $F \leftarrow NF;$ 
14      $NF \leftarrow \emptyset;$ 

```

在上述算法中,行 7、8 遍历 F 所有顶点的所有邻居,当 F 顶点较多时,搜索的邻居顶点大多已被访问过,会产生很多无效搜索,影响程序性能。为避免 F 顶点较多时 top-down 搜索产生的无效搜索,降低访存开销,Beamer 等人^[2,3]提出了 bottom-up BFS 算法,并结合二者的优势提出 top-down 与 bottom-up 相结合的混合算法。算法 2 为串行 bottom-up BFS 算法,遍历未访问顶点的邻居,如果顶点的邻居在 F 中,则更新顶点的 *visit*、*parent*,将其加入 NF ,并结束对其他邻居的遍历。

算法 2 串行 bottom-up BFS 算法

Input: $G = \langle V, E \rangle$ // the graph

root; // root vertex

Var: F ; // frontier

NF ; // next frontier

Visit; // vertex is visited

Output: *parent*; // parent map

```

1   $F \leftarrow \emptyset; NF \leftarrow \emptyset;$ 
2  for each  $v$  in  $V$ 
3     $visit[v] = 0; parent[v] = -1;$ 
4     $visit[root] = 1; F \leftarrow F \cup \{root\};$ 
5     $parent[root] = root;$ 
6    while  $F! = \emptyset$ 
7      for each  $v$  if  $visit[v] = 0$ 
8        for  $w \in E(v)$ 
9          if  $w$  is in  $F$ 
10              $visit[v] = 1; parent[v] = w;$ 
11              $NF \leftarrow NF \cup \{w\};$ 
12             break;
13      $F \leftarrow NF;$ 
14      $NF \leftarrow \emptyset;$ 

```

2.2.2 并行 BFS 算法

BFS 算法的设计是围绕硬件系统进行的,国内外对于并行 BFS 算法的研究主要集中在多线程系统、多核系统和分布式内存系统上。在多线程系统上,Bader 等人^[4]利用 Cray MTA-2 的硬件多线程和低延时同步特性,实现了很好的性能。同时,Harish 等人^[5]在 GPU 进行了并行 BFS 算法的研究实现,获得了较好的加速效果。而多核系统上 BFS 算法的研究^[6-8]主要集中在 *visit* 的原子更新优化、线程同步优化、共享队列优化方面。在分布式存储系统上实现并行 BFS 算法时,由于每个节点只负责部分顶点的更新工作,对于非本地顶点,本结点需要在本地遍历结束时将边信息发送给此顶点的拥有节点。这样,在每层同步之前,需要一个附加的 all-to-all 步骤实现节点间通信。节点间通信是多节点系统上并行 BFS 算法实现的一个重要性能瓶颈。Edmonds^[9]采用主动消息通信实现 1-D 划分算法。Cong 等人^[10]在大规模稀疏图上实现了 PGAS。Checonni 等人^[11]在 BlueGene/P 和 BlueGene/Q 系统上使用低延时的通信层来替代 MPI,对 Graph500 中的无向图搜索取得了非常好的加速效果。同时,通过维护一个已搜索顶点的前缀和来降低通信开销,避免给一个顶点多次发送父顶点更新信息。

节点间有两种数据划分方式:1-D 划分和 2-D 划分,其中,1-D 划分是针对顶点的划分,2-D 划分是针对边的划分。采用 1-D 划分,每个节点负责部分顶点并拥有与之相关联的所有边。如图 2 所示,1-D 划分有两种方式,其中 A 表示图的邻接矩阵信息, P 表示处理节点数目。图 2a 中每个节点负责部分顶点并拥有从这些顶点发出的所有边,图 2b 中每个节点负责部分顶点并拥有指向这些顶点的所有边。对于多节点系统上的 BFS 算法,若采用图 2a 划分方式,top-down 搜索时,步骤 7 可以并行执行,但每个节点拥有从负责顶点发出的所有边,搜索到的邻居有的属于其他节点,需要将不属于本节点负责的边远程发送至其他节点;bottom-up 搜索时,步骤 7 可以并行执行,每个节点遍历部分顶点的 *visit*,更新这些顶点的 NF ,同时将此部分 NF 全局通信以开展下一层的 BFS 搜索。若采用图 2b 划分方式,top-down 搜索时,每个节点遍历所有的 F ,搜索该节点负责的部分边,更新这些顶点的 *visit*、 NF ,同时将此部分 NF 全局通信以开展下一层搜索;bottom-up 搜索时,由于每个节点拥有指向负责顶点的所有边,访问的顶点可能属

于本节点也可能属于其他节点,需要将不属于本节点负责的边发送至其他节点。与 1-D 划分不同,2-D 划分则按边进行划分^[3]。将节点按照 $P=R \times C$ 方式组织成一个 R 行 C 列的节点网络。 A 按图 3 的方式划分,处理节点 $P(i,j)$ 负责 C 块数据($A_{i,j}^{(1)}, \dots, A_{i,j}^{(C)}$)。顶点 V 划分为 $R \times C$ 块,节点 $P(i,j)$ 负责第 k 块($k=(j-1) \times R+i$)。在采用 2-D 划分的 BFS 算法中,包含两个节点间数据通信阶段:扩展(Expand)和折叠(Fold)。Expand 阶段,每个节点将其 F 信息在同一个节点列间进行全局通信以组成一个新的 F ; Fold 阶段,每个节点 F 开始进行 BFS 搜索,将不属于本节点负责的边发送至其负责节点。1-D 划分是 2-D 划分的特殊形式, $R=1$ 、 $C=1$ 分别对应图 2 中的两种 1-D 划分方式。实验表明^[12],在节点规模较大时,相较于 1-D 划分方式,2-D 划分能有效减少节点间数据通信量。本文在 8 节点原型系统上开展 BFS 算法实现与研究,初步采用 1-D 数据划分,后续工作将进行基于 2-D 划分的 BFS 算法研究。

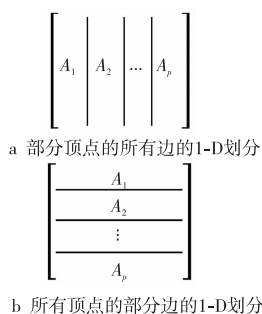


Figure 2 1-D partitioning of the parallel BFS

图 2 并行 BFS 算法 1-D 划分

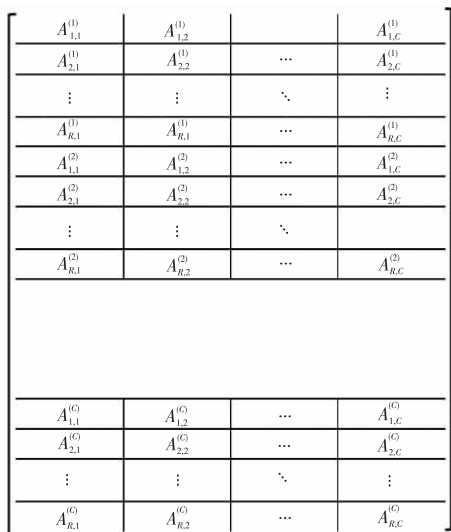


Figure 3 2-D partitioning of the parallel BFS

图 3 并行 BFS 算法 2-D 划分

3 可扩展的大数据处理并行结构原型系统

相对于传统应用,大数据处理应用访存数据量大、并行性高、访存模式不规则、程序访问时空局部性低,而传统商用处理器在处理这类应用时往往难以取得很好的性能,所以研究设计适用于大数据处理的新型计算机体系结构成为近年来研究的热点。FPGA(Field Programmable Gate Array)由于其可重构、快速开发、灵活的特性,且提供了丰富的可用资源,开始被用于高性能计算和分析领域。本课题组采用 Xilinx VC709 设计开发了适用于大数据处理的流加速器,以协处理器的方式与 Intel 商用 CPU 结合工作,并通过专用通信网络将加速器节点直接互连,设计实现了大数据处理的并行结构原型系统。

如图 4 所示,项目组设计的大数据处理系统采用异构并行的方式。计算节点 PC 的处理器采用 X86 通用处理器,负责计算任务分配、管理以及 I/O 处理等。流加速器采用 FPGA 实现,负责专门的大数据处理,计算开始时通过 PCIE 接口接收数据,处理完成后经 PCIE 接口将结果返回 PC。PC 之间通过以太网连接,实现系统管理和大规模计算作业的加载。加速器之间通过专用通信网络 InfiniBand 互连,以获取低延迟的远程网络通信。

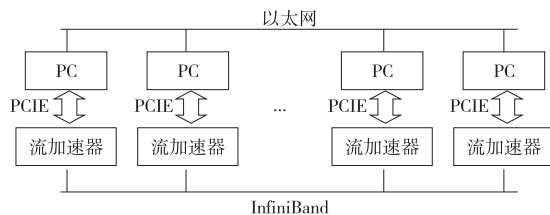


Figure 4 Interconnection structure of the parallel system

图 4 并行系统互连结构

图 5 为单节点流加速器结构框图,加速器内部有 8 个处理器核,核内采用 8 级流水设计。加速器内有 64 个硬件线程,每个处理器核有 8 个硬件线程。每个线程拥有私有的本地寄存器文件 LRF (Local Register File),大小为 0.5 KB。所有线程共享流寄存器文件 SRF (Stream Register File) 和片上共享存储 SPM (Scratch Pad Memory)。SRF 大小为 512 KB,SPM 大小为 8 KB。每个处理器核私有 0.5 KB 的一级指令缓存(L1 ICache),8 个核共享 4KB 的二级指令缓存(L2 ICache)。每个处理器使用访存指令经访存单元 LSU (Load Store

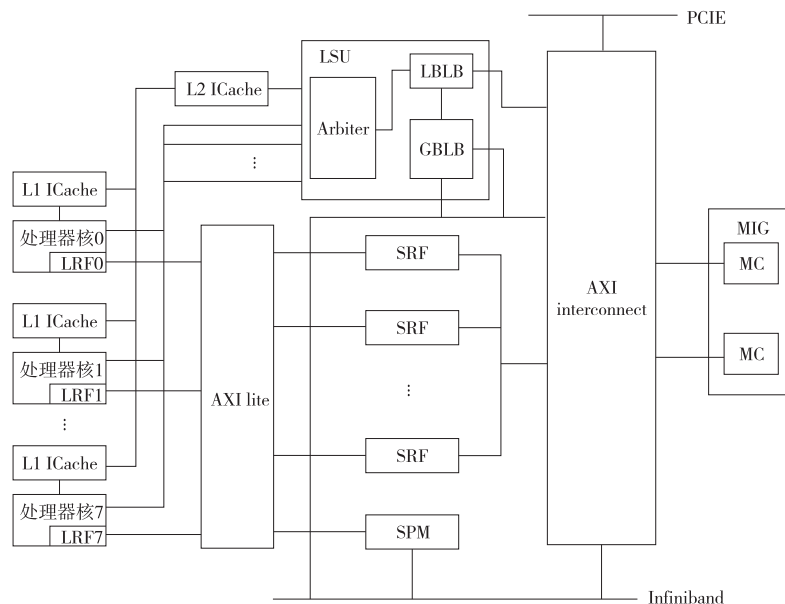


Figure 5 SAU architecture of the single node

图 5 单节点流加速器结构

Unit) 访问 DDR3。加速器通过 PCIE 和 InfiniBand 接口分别与计算机节点和其他加速器节点进行数据通信。LSU、SRF、PCIE 和 InfiniBand 通过 AXI4 交叉开关与 MIG (Memory Interface Generator) 互联, MIG 内包含两个存控 MC (Memory Controller)。

大数据应用具有指令级并行度低、线程级并行度高的特点,加速器处理器核内的 8 个线程采用交叉多线程的方式执行,可以有效开发线程级并行性,从而提升性能。交叉多线程是细粒度多线程的一种具体实现,其特点是线程交替执行的方式简单。交叉多线程要求每个时钟周期都切换线程,不论正在执行的线程是否发生阻塞,都会流向下一级流水段。这样便可确保任意时刻流水线都会充满,减少片上资源闲置。同时,交叉多线程的执行方式可以避免数据相关、降低设计的复杂度,且能有效地隐藏访存开销。大数据应用具有访存数据量大、访存模式不规则、程序访存时空局部性差的特点,传统商用处理器在处理此类应用时 Cache 失效率较高,导致处理器性能的降低和功耗的增加。为解决 Cache 利用率低的问题,项目组设计了 LRF-SRF-DDR3 三级存储层次,设计了程序员显式可控的流寄存器结构和访存方法。将处理器片内数据的组织结构和访问模式显式地暴露给程序员,由程序员结合应用程序的特征来组织数据,从而更好地适应实际应用的访存需求。

如图 5 所示,加速器节点之间通过 InfiniBand 互连组成专用的通信网络。与传统的集群系统和

高性能计算机相比,我们设计的异构并行结构有三点不同:

(1) 节点间互连方式不同:前者通常采用商用或是专门定制的网络接口卡和网络交换设备互连;后者使用比传统网络接口卡更低延时的专用网络接口,同时避免因使用 PCIE 接口的网络接口卡引起的 PCIE 总线网络延迟。

(2) 节点间通信方式不同:前者一般基于 MPI (Message Passing Interface) 进行通信,用户程序的数据包需要经过通信库、操作系统、驱动程序等多层处理,传输延迟大,且一般适用于批量消息传送;后者基于硬件机制进行通信,避免 MPI 通信的多层延迟,可适用于细粒度通信。

(3) 并行系统主存结构不同:前者通常是分布式主存结构,节点间内存不可直接访问,一般以消息传递方式进行数据通信;后者采用分布式共享主存结构,全局统一编址,每个加速器节点可以通过全局地址直接访问任意节点的内存。

4 多节点原型系统上并行 BFS 算法的设计实现

大数据问题往往涉及的数据规模较大,我们需要采用多节点系统来并行处理。BFS 算法具有计算访存比小、访存数据量大等特性,因此访存开销是影响性能的重要因素。同时,对于多节点系统上的 BFS 算法,节点间的数据通信开销也是影响程序执行性能的重要因素。因此,本文对于多节点原

型系统上并行 BFS 算法的设计主要从三个方面开展研究:合理的数据划分以充分利用数据的并行性、有效的算法设计和数据预处理以减少算法访存开销、适当的远程通信方式以减少程序远程通信开销。

4.1 1-D 数据划分设计

如前所述,2-D 的数据划分方式适用于较大规模的节点数目,本文在 8 节点的并行结构原型系统上实现 BFS 算法,将采用 1-D 的数据划分方式,基于 2-D 划分的 BFS 算法研究将在后续工作中进行。如图算法 3 所示,本文采用 top-down 和 bottom-up 相结合的混合 BFS 算法。如 2.2.2 节所述,top-down 搜索时,若采用图 2a 所示的划分方式,每个线程将独立负责部分顶点的所有边,由于此时 F 较少,搜索任务会集中在部分节点上,这样会引起严重的负载不均衡问题,导致程序的并行性差,影响算法的执行性能;若采用图 2b 所示的划分方式,每个线程负责搜索所有顶点的部分边,线程负载均衡可以很好地并行处理。bottom-up 搜索时,每个线程拥有部分顶点并负责更新其 $visit$ 、 NF 和 $parent$ 信息,若采用图 2b 所示的划分方式,每个线程拥有所有顶点的部分边,需要将不属于本线程的边发送至其负责线程,线程之间存在频繁的远程数据通信,影响数据的并行处理;若采用图 2a 的划分方式,则每个线程拥有指向其负责顶点的所有边,数据的访问只在本节点上,更有利于 bottom-up 搜索的进行。因此,本文设计的混合算法在 top-down 搜索时采用图 2b 所示的 1-D 数据划分,在 bottom-up 搜索时采用图 2a 所示的 1-D 数据划分。这样可以使 8 个加速器节点供 512 个硬件线程充分并行搜索,且使节点间的远程通信仅限于每层 NF 的全局通信。

算法 3 多节点原型系统上并行混合 BFS 算法

Input: $G=\langle V,E \rangle$ // the graph

$root$: // root vertex

$Var:F$: // frontier

NF : // next frontier

$Visit$: // Vertex is visited

Output: $parent$: // parent map

```

1  for each  $v$  in  $V$ 
2     $visit[v]=0; parent[v]=-1; F[v]=0; NF[v]=0;$ 
3     $visit[root]=1; parent[root]=root; F[root]=1;$ 
4     $level=0; endFlag=0;$ 
5    while  $endFlag \neq 1$ 
6       $endFlag=1;$ 
7      if  $level < \alpha$  // top-down step */

```

```

8      for each  $v$  in  $V$  parallel
9        if  $F[v]=1$ 
10         for  $w \in E(v)$ 
11           if  $visit[w]=0$ 
12              $visit[w]=1; parent[w]=v;$ 
13              $NF[w]=1; endFlag=0;$ 
14         else /* bottom-up step */
15         for each  $v$  in  $V$  parallel
16           if  $visit[v]=0$ 
17             for  $w \in E(v)$ 
18               if  $F[w]=1$ 
19                  $visit[v]=1; parent[v]=w; NF$ 
20                  $[v]=1; endFlag=0; break$ 
21    $F=NF;$ 
22   for each  $v$  in  $V$   $NF[v]=0;$ 
23    $level++;$ 

```

4.2 优化的并行混合 BFS 算法设计

BFS 搜索过程中存在密集的访存操作,虽然微处理器的交叉多线程机制可以隐藏部分访存延时,但访存开销仍是影响算法性能的重要因素。本文采用 top-down 和 bottom-up 相结合的混合算法,以有效减小在 F 较大时的访存开销。如表 1 所示,在输入图 G 中存在很多孤立顶点(顶点的度为零),bottom-up 搜索时存在对这些顶点的无效访问,本文在 BFS 搜索前进行数据预处理,删除孤立顶点,减小程序访存开销。在 bottom-up 搜索时,需遍历未被访问顶点的邻居顶点,若邻居顶点在 F 中,则更新顶点的 $visit$ 、 $parent$ 、 NF 并结束遍历。bottom-up 搜索时若能一次命中(一次查询即可找到顶点的父亲顶点),则可减少对邻居信息的访存操作。Yuichiro 等人^[6]认为顶点的度越大,其在 F 中的概率就越大。本文在 BFS 搜索前进行数据预处理,将顶点的邻居顶点按度的大小进行降序排列,以提升 bottom-up 搜索的一次命中率,减小访存开销。BFS 搜索中存在频繁的数据访问,本文在设计中采用细粒度的本地访存和远程访存,减少无效的访存操作,以提升性能。

4.3 远程通信方式设计

节点间的远程通信对多节点并行 BFS 算法性能有着重要影响。本文采用基于 1-D 划分的并行 BFS 算法,节点间远程通信仅限于每层 NF 的全局通信。在本文的 BFS 算法设计中,top-down 搜索时,因 NF 较少,采用分散通信的方式,仅当 NF 有更新时进行全局通信;bottom-up 搜索时,next frontier 连续访问,采用集中通信的方式,一次读出的 512 比特搜索完成后才进行通信,同时设置更

新有效位,仅在 NF 更新有效时远程通信。本文对于 top-down 和 bottom-up 搜索过程根据 NF 访问方式和访问数据量的不同采用不同的通信方式,减少远程通信的数据量,以提升性能。

Table 1 Number and ratio of isolated vertexes in the graphs of different scales

表 1 不同规模的图中孤立顶点数和比例

count	percent/%
188 751	36.0
851 710	40.6
3 777 749	45.0

5 多节点原型系统上并行 BFS 算法的性能测试与分析

5.1 实验环境

大数据处理原型系统采用 Verilog 编写,开发环境为 Vivado 2013.4,验证环境为 Xilinx VC 709 开发板平台。在 Verilog 程序通过综合、布局布线、生成二进制文件之后,将生成的硬件结构写入开发板,完成开发板的硬件配置。在 Linux 环境下,将 BFS 算法程序和实验数据通过 PCIe 接口写入加速卡上的 DRAM,在数据处理完成之后,再经 PCIe 接口返回主机,完成数据正确性验证。实验通过在硬件程序中加入 ila 核探针来检测加速器的数据处理时间,作为计算系统处理性能的指标。

5.2 性能测试分析

本文首先在 2^{19} 顶点、 2^{23} 边的数据规模下进行 BFS 算法实现和性能测试分析,并进一步在 $2^{20} \sim 2^{22}$ 顶点规模下进行性能测试(如图 6 所示)。表 2 为 2^{19} 规模各层顶点数目(BFS 前两层采用 top-down 搜索,后面各层采用 bottom-up 搜索)。

如图 6 所示,本文在并行结构原型系统上实现了基于 1-D 划分的混合 BFS 算法,并在此基础上进行删除孤立顶点优化,避免 bottom-up 过程中对孤立顶点的无效搜索,为减少 bottom-up 过程对边的无效搜索,本文在上述优化的基础上对顶点的边进行度排序,性能提升了 13.8%。

Table 2 Number of vertexes under the scale of 2^{19}

表 2 2^{19} 数据规模时各层顶点数目

0	1	2	3	4	5
1	2 507	225 697	106 371	784	2

本文在多节点 BFS 算法实现的基础上,对 BFS 搜索的 top-down 和 bottom-up 过程进行计

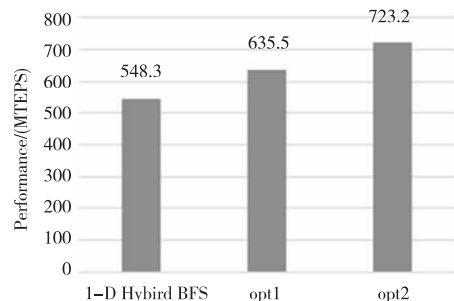


Figure 6 Performance of BFS under the scale of 2^{19}

图 6 2^{19} 数据规模 BFS 性能测试

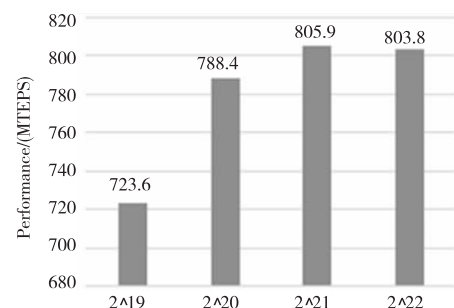


Figure 7 Performance of BFS under the scale of $2^{19} \sim 2^{22}$

图 7 $2^{19} \sim 2^{20}$ 数据规模时 BFS 性能测试

时,计算两个搜索过程的占比,如图 8 所示, top-down 过程虽然搜索顶点较少,但仍占时较多,分析 top-down 过程中每个线程需要搜索所有的 F ,存在较多的本地访存和无效搜索,后续工作中可针对 top-down 过程进行优化。

多节点系统 BFS 算法中节点间数据通信是影响性能的重要因素。本文在基于 1-D 划分的 BFS 算法中测试节点间通信时间,如图 9 所示,在 8 节点原型系统上,节点间数据通信时间为程序执行总时间的 4.3%,后续工作可对节点间的数据通信进行优化,但优化效果可能有限。

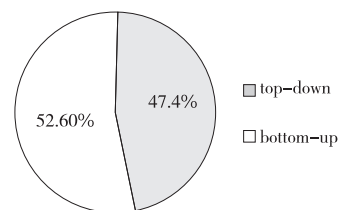


Figure 8 Time percentage of top-down and bottom-up

图 8 top-down bottom-up 过程占比

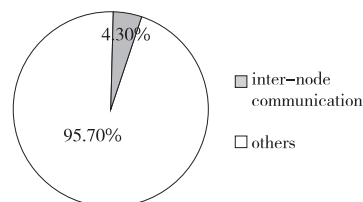


Figure 9 Time percentage of inter-node communication

图 9 节点间全局通信(NF)占比

6 结束语

本文对 Graph500 基准测试和 BFS 算法进行介绍,并介绍了课题组设计的大数据处理并行结构原型系统,同时在多节点原型系统上设计实现了并行 BFS 算法。本文从开发 BFS 算法并行性、降低算法访存开销和降低节点间通信开销三个方面开展研究,在 8 节点并行处理原型系统上设计实现了基于 1-D 划分的混合 BFS 算法,并实现了删除孤立顶点和度排序的优化,减少 bottom-up 过程的无效访存,提升性能。本文在 2^{22} 顶点、 2^{26} 边的数据规模下进行性能测试分析,取得了 803.8 MTEPS 的处理性能。同时,本文在算法实现的基础上,测试了 BFS 算法 top-down 和 bottom-up 过程的占比和节点间数据通信时间,分析后续工作可针对 top-down 过程进行优化,减少此过程的无效访存,提升性能。

参考文献:

- [1] www.graph500.org.
- [2] Beamer S, Asanovic K, Patterson D A. Searching for a parent instead of fighting over children: A fast breadth-first search implementation for graph500; Tech Rep UCB/EECS-2011-117[R]. Berkeley; EECS Department, University of California, 2011.
- [3] Beamer S, Asanovic K, Patterson D. Direction-optimizing breadth-first search[J]. Scientific Programming, 2013, 21 (3-4): 137-148.
- [4] Bader D, Madduri K. Designing multithreaded algorithms for breadth-first search and st-connectivity on the Cray MTA-2 [C]//Proc of the 35th International Conference on Parallel Processing (ICPP), 2006:523-530.
- [5] Harish P, Narayanan P J. Accelerating large graph algorithms on the GPU using CUDA [C]//Proc of High Performance Computing (HiPC 2007), 2007:197-208.
- [6] Agarwal V, Petrin F, Pasetto D, et al. Scalable graph exploration on multicore processors[C]//Proc of ACM/IEEE International Conference on High Performance Computing, Networking, Storage and Analysis (SC 2010), 2010:1-11.
- [7] Xia Y, Prasanna V. Topologically adaptive parallel breadth-first search on multicore processors[C]//Proc of International Conference on Parallel and Distributed Computing Systems (PDCS), 2009:1.
- [8] Leserson C E, Schardl T B. A work-efficient parallel breadth-first search algorithm[C]//Proc of SPAA 10, 2010: 303-314.
- [9] Edmonds N, Willcock J, Hoeftler T, et al. Design of a large-scale hybrid-parallel graph library[C]//Proc of International Conference on High Performance Computing, Student Research Symposium, 2010:1.
- [10] Cong G, Almasi G, Saraswat V. Fast PGAS implementation of distributed graph algorithms[C]//Proc of International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2010), 2010:1-11.
- [11] Checconi F, Petrini F, Willcock J, et al. Breaking the speed and scalability barriers for graph exploration on distributed-memory machines[C]//Proc of 2012 International Conference on High Performance Computing, Networking, Storage and Analysis (SC 2012), 2012: 1-12.
- [12] Ueno K, Suzumura T. Highly scalable graph search for the graph500 benchmark[C]//Proc of the 21st ACM International Symposium on High-Performance Parallel and Distributed Computing, 2012: 149-160.
- [13] Yasui Y, Fujisawa K, Sato Y. Fast and energy-efficient breadth-first search on a single NUMA system[C]//Proc of International on Conference on Supercomputing, 2014: 365-381.

作者简介:



衡冬冬(1991-),男,河南淅川人,硕士生,研究方向为计算机系统结构。E-mail: hengdd1991@163.com

HENG Dong-dong, born in 1991, MS candidate, his research interest includes computer architecture.



唐玉华(1962-),女,研究员,研究方向为计算机系统结构。E-mail: yhtang@163.com

TANG Yu-hua, born in 1962, research fellow, her research interest includes computer architecture.