计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.191000013

面向国产异构众核处理器SW26010的BFS优化方法

袁欣辉　林蓉芬　魏　迪　尹万旺　徐金秀
江南计算技术研究所　江苏 无锡 214083

摘　要　近年来,人们越来越关注计算机对数据密集型课题的处理能力。宽度优先搜索(Breadth First Search,BFS)是一种典型的数据密集型课题,被广泛应用于多种图算法。Graph 500 Benchmark以BFS搜索为核心算法,已经成为评价计算机处理大数据能力的基准。神威太湖之光超级计算机从2016年6月至2017年11月连续4次荣登Top 500榜单榜首,其处理器SW26010是首款由我国自主研制的异构众核处理器。文中研究了如何利用SW26010的体系结构特点加速BFS算法的问题,在SW26010上实现了基于单个核组的方向优化的融合BFS算法,使用字节图(bytemap)释放内层循环依赖性,利用异步DMA隐藏计算与便签存储器的访问开销,利用异构架构协同运算并对图做预处理。最终,以Graph 500作为基准测试程序处理scale为22的图,SW26010处理器单核组BFS的性能达到457.54MTEPS。

关键词:SW26010;神威太湖之光;Graph 500;数据密集;异构众核;宽度优先搜索
中图法分类号　TP311

Optimization of BFS on Domestic Heterogeneous Many-core Processor SW26010

YUAN Xin-hui,LIN Rong-fen,WEI Di,YIN Wan-wang and XU Jin-xiu
Jiangnan Institute of Computing Technology,Wuxi,Jiangsu 214083,China

Abstract　In recent years,there is growing concern for the processing capabilities of data-intensive task. Breadth-first search (BFS) is a typical data-intensive problem,which is widely used in a variety of graph algorithms. Graph 500 Benchmark,taking BFS algorithm as the core,has become the benchmark for the evaluation of processing capabilities of data-intensive tasks. Sunway TaihuLight supercomputer topped the Top 500 list for four consecutive times from June 2016 to November 2017,the processor of which,named SW26010,is the first Chinese homegrown heterogeneous many-core processor. This paper studies how to use the architecture characteristics of SW26010 to accelerate BFS algorithm. A direction-optimizing hybrid BFS algorithm based on a single core group(CG) is implemented on SW26010, using bytemap to release the data dependencies in inner loops,hiding overhead of calculation and SPM access by using asynchronous DMA,taking advantage of heterogeneous architecture to compute collaboratively and carrying out graph preprocessing. Eventually,with Graph 500 as the benchmark processing a scale 22 graph,a single CG of SW26010 processor achieves a performance of 457.54MTEPS.

Keywords　SW26010,Sunway TaihuLight,Graph 500,Data-intensive,Heterogeneous many-core,Breadth First Search

随着大数据的兴起,数据密集型应用越来越得到人们的重视。目前,一项重要挑战是:面对超大规模、低局部性的数据集,如何快速处理并发现各个数据之间的联系。图是一种能够很好地描述这类问题的数据结构,大规模图的分析是社交网络和消费偏好分析等很多领域的热点课题。BFS算法是许多常用图分析算法的核心子算法,如最短路径、最小费用最大流等。Graph 500 Benchmark[1]使用BFS算法处理一类经Kronecker生成器生成的随机图,随机指定64个根,按照宽度优先生成树,分别计算每秒钟遍历的边数(Traversed Edges Per Second),并以这64个结果的调和平均数作为排序指标。目前,Graph 500已经成为评价计算机处理数据密集型应用的基准。

1　相关工作及BFS算法的简要介绍

伴随大数据研究的深入,研究人员从算法和体系结构两方面提出了许多BFS优化方法。

Agarwal等[2]使用比特图代表BFS中的顶点,缩小了工作集的规模,现被广泛应用于诸多BFS的实现。Ueno等[3]采用将顶点按度数降序排列的方式提高Cache命中率。Tithi等[4]采用中心任务队列或随机任务抓取的方式,不使用锁和

到稿日期:2019-10-08　返修日期:2020-06-18　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发计划资助项目(2016YFB0201100,2017YFB0202702);国家"973"计划资助项目(2014CB744100);国家"863"计划资助项目(2012AA01A306)
This work was supported by the National Key Research and Development Project of China(2016YFB0201100,2017YFB0202702),National Basic Research Program of China(2014CB744100) and National High Technology Research and Development Program of China (2012AA01A306).
通信作者:袁欣辉(yuanxh07@foxmail.com)

原子指令实现负载平衡，通过冲突检测、数据覆盖等方法保持算法的正确性。Chhugani 等[5]使用 VIS 数据结构持续追踪已经访问过的顶点，提出了一种保证负载平衡的任务划分方法，并通过数据压缩、延迟隐藏和尽量减少 BFS 算法中锁的使用及开销等手段来提升性能。

按照搜索的方向，BFS 可分为自顶而下（Topdown，TD）和自底而上（Bottomup，BU）两类。Beamer 等[6]将两种算法融合，提出了一种方向优化的 BFS 算法，减少了不必要的边的访问。Yasui 等[7]使用方向优化的融合算法，通过控制绑定计算资源使 BFS 适应于 NUMA 架构，并使用列优先划分平衡负载。Yasui 等[8]通过按度数降序排列邻接列表、剔除度数为 0 的顶点、剩余顶点重构图等预处理手段突破 BU 算法的性能瓶颈，优先搜索度数最大的若干顶点，显著提高了算法性能。

Tao 等[9]在 Mic 处理器上提出了一种松弛 BFS 算法，解决了 TD 算法多核心修改 bitmap 的竞争问题，降低了内层循环的并行依赖，并使用掩码向量加速 BFS。Busato 等[10]总结了目前 GPU 上的 BFS 研究成果，并通过动态虚拟 Warp、动态并行等方法改善负载不平衡问题。Zou 等[11]在 CPU-GPU 混合架构上实现了方向优化的 BFS 融合算法，其充分利用 CPU 和 GPU 的计算能力，根据任务动态选择最适合的执行框架：CPU 上串行 TD，CPU 上并行 TD，CPU 和 GPU 协同 BU。

此外，Checooni 等[12]曾在 Blue Gene 上使用一种基于 2D 分布和折叠翻转映射的 BFS 算法。

经典的分层次并行 BFS 算法的描述如算法 1 和算法 2 所示。

算法 1 Level-synchronized Parallel topdown-BFS

```
Function topdown-step(frontier,next,bfs_tree)
  for u∈frontier do
    for v∈neighbors(u) do
      if bfs_tree(v) ==-1 then
        bfs_tree(v)=u
        next=next∪{v}
  frontier=next
End Function topdown-step
```

算法 2 Level-synchronized Parallel bottomup-BFS

```
Function bottomup-step(frontier,next,bfs_tree)
  for v∈ertices do
    if bfs_tree(v)==-1 then
      for u∈neighbors(v) do
        if u∈frontier then
          bfs_tree(v)=u
          next=next∪{v}
          break
  frontier=next
End Function bottomup-step
```

本文尝试在国产异构众核处理器 SW26010 上加速 BFS 算法，结合体系结构特点实现和优化了上述 BFS 算法，并采用字节图（bytemap）、异构协同、图的预处理等方法来提高性能。同时，对各优化手段在 SW26010 单核组上取得的效果进行了实验分析，为在异构众核处理器上高效实现 BFS 做了有益的尝试。

2 国产异构众核处理器 SW26010

SW26010 是我国自主研发的首款异构众核处理器，其基本结构如图 1 所示。SW26010 采用类似 Host＋Device 的片上异构架构，由 4 个核组（Core Group，CG）组成，每个核组包含一个管理核心（Management Processing Element，MPE）和 64 个运算核心（Computing Processing Element，CPE），4 个核组由片上网络（Network on Chip，NoC）互连。每个核组拥有自己的内存空间，内存通过存储控制器（Memory Controller，MC）与管理核心和运算核心簇连接。

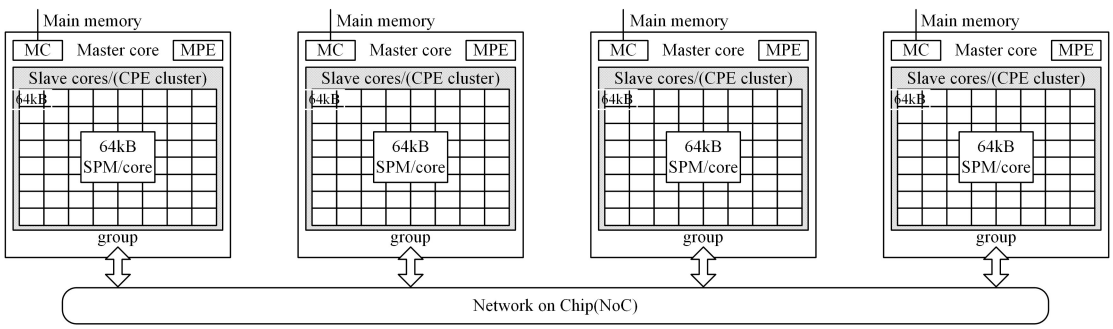

图 1 SW26010 的基本结构[13]

Fig. 1 General architecture of SW26010[13]

管理核心，也称作主核，为具有完整功能的 64 位 RISC 结构通用核心，支持中断、内存管理、256 bit 超标量运算、乱序执行等，拥有 32 kB L1 数据 Cache、32 kB L1 指令 Cache 和 256 kB L2 指令以及数据共享 Cache。运算核心，也称作从核（其结构如图 2 所示），为 64 位 RISC 结构精简核心，支持乱序执行、256bit 超标量运算等，但不支持中断。每个运算核心包含一个 16 kB 的 L1 指令 Cache 和 64kB 的局部存储，其局部存储可以配置为由软件显式管理的便签式存储器（Scratch-pad Memory，SPM）。SPM 与主存之间支持双向带跨步的异步直接存储访问（Direct Memory Access，DMA）。64 个运算核心按照图 2 所示方式，以 8×8 的二维 Mesh 形式组织，同行或同列的运算核心可以通过低延迟的快速寄存器通信的方式交换数据。MPE 与 CPE 协同完成计算任务，通常 MPE 承担任务分配、核组管理与通信的任务，CPE 簇承担计算任务。

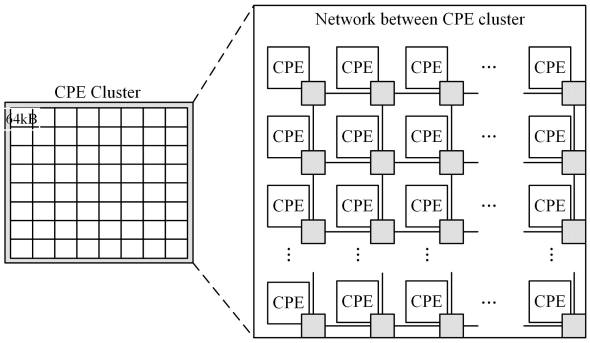

图 2 SW26010 运算的核心簇结构

Fig. 2 General architecture of CPE cluster

3 SW26010 单核组上的 BFS 优化

3.1 SW26010 体系结构的相关优化

3.1.1 数据的空间分布方式

为提高效率，本文研究过程中将从核私有数据存储配置为 SPM。基于 SW26010 实现的 BFS 采用 CSR 格式表示顶点的邻接列表，按照 1D 数据分布，将所有顶点均分在 64 个从核上，并采用比特图(bitmap)与字节图(bytemap)结合的方式来表示 *In*，*Out* 和 *Vis* 数组。其中，*In* 数组记录当前 BFS 搜索轮次的父亲顶点，其初始值为整个图的根；*Out* 数组记录当前搜索轮次找到的孩子顶点；*Vis* 数组记录当前搜索轮次之前已经访问过的顶点。bitmap 的每一位(bytemap 的每个字节)在 3 个数组中分别表示某个顶点是否是当前层次的父亲顶点、新找到的孩子顶点以及已经访问过的顶点。在各从核的 SPM 中，使用 bitmap 表示该从核分配到的顶点的 *In*，*Out* 和 *Vis* 数组；在主存中，使用 bitmap 表示全局图的 *In* 数组，用于 BU 算法获取全局信息，并使用 bytemap 表示 TD 算法中的 *Out* 数组；此外，邻接列表和父亲树 *bfs_tree* 被置于主存中。

这种数据分布方式充分考虑了 SW26010 架构的特性与 BFS 算法的特点。

(1)从核访问 SPM 的速度远快于直接访问主存。*In*，*Out* 和 *Vis* 是 BFS 需要频繁访问的数组，放置在 SPM 中可以提高各从核的搜索速度；使用 bitmap 表示数组可以降低 SPM 的空间开销，使之能适应更大规模的图的处理；此外，使用 bitmap 还可以降低从核 DMA 的数据量，进而提高效率。

(2)BU 算法天然具有并发性，非常适合 SW26010 的众核架构。BU 算法中各核心须获取全局的 *In* 数组信息，但只需要修改本地的局部 *Out* 数组。基于该特点，BU 算法可采用 bitmap 形式表示 *In*/*Out* 数组，在各从核私有的 SPM 中存放 *Out* 数组，并将全局 *In* 数组放置于核组共享的主存中。利用上述数据组织方式实现的 BU 算法具有空间开销小、DMA 数据量少且不需要使用锁的优点，有助于算法的高效并行。

(3)CSR 格式下，TD 算法需要从核阵列写共享的 *Out* 数组，使用 bytemap 表示 *Out* 数组能够避免锁的使用，进而提高各从核内层循环的并发性。

3.1.2 使用 DMA 优化 BFS

使用 DMA 可以提高运算核心访问主存连续数据段的效率，但会引入 DMA 启动开销。SW26010 支持异步 DMA，这为算法实现中隐藏 SPM 访问与计算开销(主要是 bitmap 的清零和逻辑运算)提供了可能。

设置 DMA 数据量的下限值 *OPT_DMA_BLOCK*，当邻接列表的数据量超过该值时，使用 DMA 分段读入；否则各从核直接访问主存。这种方式能够提高访存效率，同时避免对过小数据 DMA 带来的性能下降问题。另外，对 BU 算法还应设置数据量上限，分段 DMA 读入：如果某个未被访问过的顶点找到了一个在 Frontier 中的父亲，那么该顶点的 BU BFS 搜索结束，将排在该父亲顶点后面的邻接列表通过 DMA 传输到 SPM 中是没有意义的。

单个处理器的 BFS 性能主要受限于访存带宽，因此应尽可能地将计算等时间隐藏于从核的 DMA 中。SPM 中的 *In*，*Out* 和 *Vis* 数组须频繁进行清零和逻辑运算，可以利用异步 DMA 隐藏从核计算和访问 SPM 的开销，使得 BFS 的运行时间几乎等于 DMA 与离散访存时间的加和。此外，可以使用 SIMD 加速清零与逻辑运算，避免 DMA 隐藏不住而出现露头开销。

3.1.3 使用寄存器通信优化运算核心簇的信息交互

运算核心簇需要计算各层次搜索到的新顶点总数 *Out_num* 和新顶点的邻边总数 *Edge_num*，并将其作为判断 BFS 是否完成以及 TD 和 BU 算法切换的判据。为得到这两个值，可以使用主存原子加，但会引入核组串行访主存的开销。SW26010 上的 fetch_and_add 由软件锁实现，开销可近似表示为 $L=2*Num_{core}*Latency$；也可以通过各运算核心并发写主存不同位置，然后由管理核心完成计算，但需多次进行管理核心函数与运算核心函数的切换，引入了 spawn-join 的并行开销。

SW26010 上同行或同列的运算核心每次可以通过寄存器通信交换 32B 的数据，使用寄存器通信可以在几十拍内完成各核心数据的规约求和，且不必在主核函数与从核函数间切换。

3.2 算法的相关优化

3.2.1 topdown 算法的实现

使用 bitmap 可以压缩 BFS 算法中 *In*，*Out* 和 *Vis* 数组的存储空间，进而增加这些数据的局部性；还能并行判断 64 个顶点是否都在 Frontier 中或都没有被访问过，从而降低循环开销。但对于 SW26010 这种众核架构，CSR 格式下，使用 bitmap 会在 TD 算法中产生竞争问题：1)如果将 bitmap 放在从核阵列中，须写其他从核的 SPM，目前硬件不支持 RDMA，而寄存器通信必须收发双方显式配对，不适合 TD 算法的需求。2) 如果将 bitmap 放在主存中，从核阵列内多个核心有可能同时试图改写 *Out* 数组，进而产生“读-修改-写”竞争。若两个从核恰好同时试图将 *Out* 中属于同一字节的两个不同比特位置 1，则后完成的写操作会覆盖稍前完成的操作，造成某些比特位置 1 失败。如图 3 所示，$Byte_k$ 的正确值应当是 0x84，但当竞争出现时，$Byte_k$ 的值变为 0x04 或 0x80。因此，为保证结果的正确性，TD 算法对 *Out* 数组执行置 1 操作时必须使用锁，这将导致 TD 算法内层循环“串行化”，效率较低。

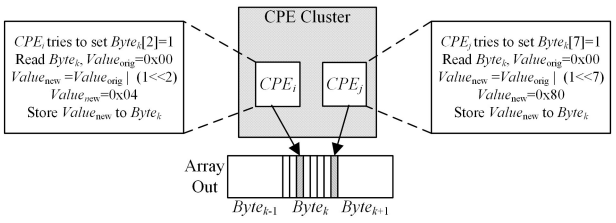

图 3 *Out* 数组竞争示例

Fig. 3 Racing example of array *Out*

为避免使用锁，在 SW26010 上设计实现了两种 topdown 算法：利用 bitmap 结合父亲树中特殊标记的 tag-TD 算法（算法 3）和利用 bytemap 表示主存中 *Out* 数组的 bytemap-TD 算法（算法 4）。

算法 3 Tag-Top-Down BFS Algorithm

```
Function topdown-step(In,Out,Vis,bfs_tree)
1. for i=vtx_start/64 to vtx_end/64 do
2.     if In[i] ==0 then continue
3.     for j=0 to 63 do
4.       if In[i]&j ==1 then
5.         u=vtx_start+64*i+j
6.         dma CSR-Format Edge list of vertex u and wait dma done
7.         for vertex v∈neighbours of u do
8.           if bfs_tree(v) ==-1 then
9.             bfs_tree(v)=u-num_of_vtx_all-1
10. array barrier
11. dma bfs_tree to SPM
12. set Out=Ø
13. wait dma done
14. for i=vtx_start to vtx_end do
15.   if bfs_tree(i)<-1 then
16.     Out_num++
17.     bfs_tree(i)+=(num_of_vtx_all+1)
18.     set_bitmap(Out,i)
19.     front_edge_num+=number of neighbors of vertex i
20. dma bfs_tree to memory
21. bitadd Out to Vis
22. wait dma done
23. reduce(Out_num,'ADD')
24. reduce(front_edge_num,'ADD')
25. swap In and Out
```

tag-TD 算法中，顶点编号介于[0，$num\_of\_vtx\_all-1$]。第 9 行将新找到的顶点 v 的父亲值设为一个小于 −1 的值 u-$num\_of\_vtx\_all-1$。第 11−13 行将父亲树 bfs_tree 通过异步 DMA 读入 SPM 中，同时完成对 *Out* 数组的清零。第 14−19 行，如果 $bfs\_tree(i)$ 的值小于 −1，表明该顶点是本轮新找到的顶点，将用 bitmap 表示的 *Out* 数组中的对应位置置 1。第 19 行与第 21 行计算新顶点邻边的总数，并更新 *Vis* 数组，用于切换 BU 算法。

tag-TD 算法的第 9 行也会出现竞争，但硬件保证了其正确性，因此无须使用锁。尽管该算法增加了一些额外的访存开销（第 11−13 行和第 20−22 行），但增加的开销是连续访存。先标记 bfs_tree 再更新 *Out* 数组使得 *Out* 可以放置在 SPM 中，且不会发生 bitmap 竞争，提高了内层循环的并发度，总体上提高了算法性能。

算法 4 Byte-Top-Down BFS Algorithm

```
Function topdown-step(In,Out_char,Vis,bfs_tree)
1. for i=vtx_start/64 to vtx_end/64 do
2.     if In[i] ==0 then continue
3.     for j=0 to 63 do
4.       if In[i]&j ==1 then
5.         u=vtx_start+64*i+j
6.         dma CSR-Format Edge list of vertex u and wait dma done
7.         for vertex v∈neighbours of u do
8.           if bfs_tree(v) ==-1 then
9.             bfs_tree(v)=u
10.            Out_char(v)=1
11. array barrier
12. dma Out_char to SPM
13. set Out=Ø
14. wait dma done
15. for i=vtx_start to vtx_end do
16.   if Out_char(i) ==1 then
17.     Out_num++
18.     set_bitmap(Out,i)
19.     front_edge_num+=number of neighbors of vertex i
20. bitadd Out to Vis
21. memset Out_char[vtx_start…vtx_end] to zero using dma
22. reduce(Out_num,'ADD')
23. reduce(front_edge_num,'ADD')
24. swap In and Out
```

byte-TD 算法与 tag-TD 算法总体架构类似，不同之处是其第 10 行找到新顶点后设置主存中的 bytemap 数组 *Out_char*，由于 *Out_char* 是 char 类型的数组，因此当竞争出现时算法的正确性由硬件保证，不需使用锁；*Out_char* 为 char 型，而 bfs_tree 为 unsigned long 型，因此第 12 行与第 21 行的 DMA 数据量缩减为 tag-TD 算法的 1/8，但主存中 *Out* 数组所需空间变为 8 倍。

如图 4 所示，byte-TD 算法的整体性能比 tag-TD 算法略低。当 Frontier 中的顶点较少时，连续访存时间占比较高，由于 byte-TD 算法的 DMA 数据量更少，因此其效率更高；但当 Frontier 中顶点较多时，byte-TD 算法第 10 行中带宽争用明显，且 byte-TD 算法在这些层次引入的离散访存开销超过了在其他层次的收益，因此其性能略低。

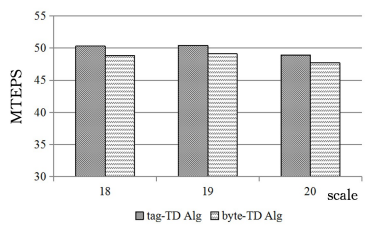

图 4 不同 scale 下两种 TD 算法性能的比较

Fig. 4 Comparison of two TDs in different scales

TD-BU 融合算法中，TD 算法主要用于 Frontier 顶点较少的算法启动和临近结束阶段的 BFS 搜索。规模 $scale=18$

时，两种算法的时间开销分布如表1所列。可以看到，BFS启动和临近结束的几个层次中，Frontier规模较小，byte-TD算法的性能明显优于tag-TD算法。因此，在方向优化的融合BFS算法中，TD算法采用byte-TD实现。

表1 两种TD算法的拍数分布统计

Table 1 Clock statics of two TD Algorithms

BFS 树层次	Frontier 中顶点数	tag-TD 算法			byte-TD 算法		
		总拍数	Search	Set bitmap	总拍数	Search	Set bitmap
0	1	417772	31066	386706	134394	30648	103746
1	4	938968	479199	459769	708333	586733	121600
2	2898	32307764	29813673	2494091	35460625	33355817	2104808
3	132566	60320504	59253116	1067388	61089211	60394050	695161
4	38201	1845328	1446591	398737	1558218	1451834	106384
5	226	482766	96395	386371	204347	100164	104183
6	1	459017	71498	387519	183254	79240	104014

3.2.2 BU算法的实现

BU算法中*In*，*Out*和*Vis*等数据均采用bitmap表示，其具体步骤如算法5所示。

算法5 Bottom-Up BFS Algorithm

```
Function bottomup-step(Out_curr,Out_next,Vis,In_memory_curr,
In_memory_next,bfs_tree)
1. for i=vtx_start/64 to vtx_end/64 do
2.     if Vis[i] ==0xffffffffffffffffL then continue
3.     for j=0 to 63 do
4.       if Vis[i]&j ==0 then //have not visited yet
5.         v=vtx_start+64*i+j
6.         for vertex u∈neighbours of v do
7.           if u is in frontier In_memory_curr then
8.             bfs_tree(v)=u
9.             set_bitmap(Out_curr,v)
10.            Out_num++
11.            break
12. dma Out_curr to memory In_memory_next
13. set Out_next=Ø
14. bitadd Out_curr to Vis
15. reduce(Out_num,'ADD')
16. wait dma done
17. swap In_memory_curr and In_memory_next
18. swap Out_curr and Out_next
```

BU算法第6行读取顶点v的邻边列表。正如前面指出的，一旦v找到某个父亲顶点u位于当前Frontier，则对v的搜索应立即结束(第11行)，因此将v的邻边使用DMA全部读入SPM是不划算的，实现中第6行直接访问主存而不使用DMA。第7行通过直接读取主存中的bitmap数组*In_memory_curr*判断顶点u是否位于当前的Frontier中；各从核须将本轮的搜索结果*Out_curr*通过DMA传输到主存*In_memory_next*中，并在本轮结束后将*In_memory_curr*与*In_memory_next*交换。可利用DMA传输*Out_curr*的时间来隐藏bitmap的清零、bitadd等运算。

3.2.3 方向优化的TD-BU融合算法

正如文献[6]中描述的，TD与BU算法的执行时间都近似由各层需要遍历的边数决定。TD算法需要遍历Frontier中每个顶点的每一条邻边；而BU算法需要遍历未被访问过的顶点的邻边，但当找到某个邻居位于Frontier中时，遍历便可结束。随着Frontier规模的增大，TD算法的开销增大，当BFS搜索快要完成时，TD算法的开销也随Frontier的缩小而缩减；当Frontier规模达到最大时，TD算法的性能最差，而BU算法的性能最好；当BFS要结束时，Frontier越来越小，而BU算法要遍历所有未访问的点的邻边(包括与根root不在一个子图的顶点)，性能通常不及TD算法。因此，方向优化的BFS融合算法通常包括TD->BU->TD的转换过程，转换条件如图5所示。其中，*Edge_front*表示与Frontier相连的、当前需要遍历的所有边的总数，*edge_left*为到当前搜索层次还未被搜索过的点的邻边总数；*Out_num*表示Frontier中的顶点数目，*num_vtx_all*为图中所有顶点的总数。

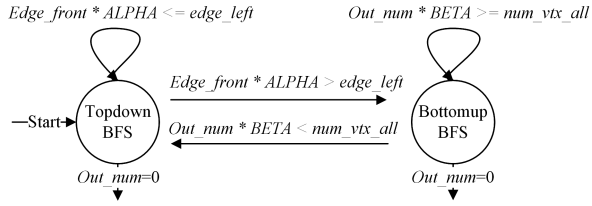

图5 TD与BU融合算法的切换条件

Fig. 5 Switch condition of hybrid algorithm

参数ALPHA与BETA对融合算法的性能有着显著影响。如图6所示，在本文实验参数配置下，对于SW26010上的TD-BU融合算法，ALPHA取15、BETA取45比较合适。

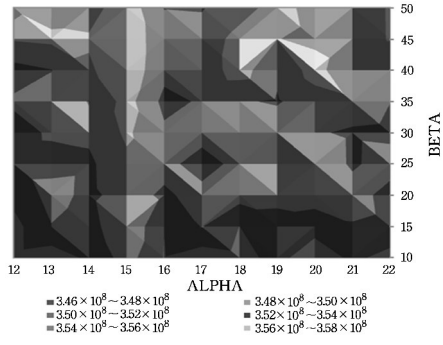

图6 *scale*=18，ALPHA-BETA-TEPS关系图

Fig. 6 ALPHA-BETA-TEPS relation graph when *scale*=18

3.3 顶点顺序预处理

在SW26010单核组上实现的BFS算法使用CSR格式表示每个顶点的邻接列表。邻接列表中顶点的排列顺序显著影响着融合BFS算法的效率：对于TD算法而言，无论邻接列表如何排列，它需要遍历的边数都等于当前Frontier的所有邻边之和，因此排列顺序对TD算法的影响不大；但是对于BU

算法，某个顶点 v 只要找到一个顶点 u 在 Frontier 中即可停止对 v 的搜索，度数高的顶点邻边多，显然更容易被搜索到而出现在 Frontier 中，因此邻接列表中顶点按度数降序排列很有可能减少 BU 算法遍历的邻边数。邻点排列顺序对算法性能的影响如图 7 所示，2－4 层切换为 BU 算法，其余层次是 TD 算法。从图中可见，降序排列对 TD 算法几乎没有影响，但与随机排列相比，邻点按度数降序排列使得 BU 算法的首次搜索时间（也是 BFS 时间的主要部分）显著下降。

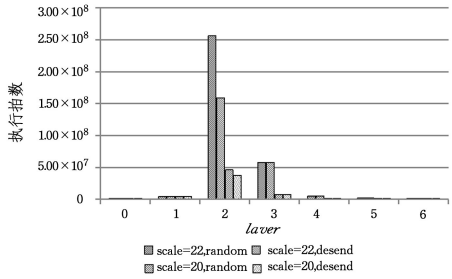

图 7 不同顶点排列顺序对 TD 和 BU 算法性能的影响

Fig. 7 Performances of TD & BU in different vertex orders

BU 算法的开销与 unvisited 的顶点数呈正相关。由 Kronecker 生成器生成的随机图中存在大量度数为 0 的点（40% 左右），避免对这部分点的遍历可以提高 BU 算法的效率。为达到该目的，我们使用一种简单易行，不同于 Yasui[8] 重设顶点号、重生成邻边列表的方法：在 BFS 初始化阶段将度数为 0 的点在 *Vis* 数组中置 1，仿佛这些点已经被访问过了，以避免在 BU 算法中重复处理这些顶点。

3.4 异构架构的主从协同优化

SW26010 是一种异构众核处理器。虽然由运算核心阵列实现的 BFS 已经将访存带宽利用得比较充分，但管理核心两级 Cache 的存在使得管理核心的部分访存需求可以直接由 Cache 满足，而不需真正访问主存。如图 8 所示，访问主存的总带宽不变，但管理核心总有些数据会命中 Cache，使得管理核心的实际可用带宽高于主存到管理核心 Cache 的访存带宽，这意味着核组异构协同运算可能获得超过主存带宽的访存性能，使得 BFS 的性能进一步得到提高。

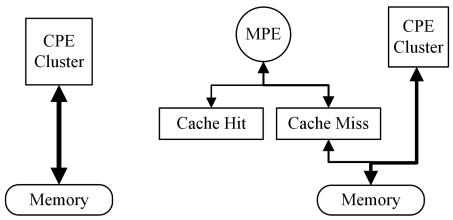

图 8 核组协同运算示意图

Fig. 8 Illustration of CG compute collaboratively

核组异构协同方法的有效性很大程度上依赖于主核访存命中 Cache。最差情况下，主核全部 Cache 不命中，则核组异构协同引入了 Cache Miss 的开销，其性能会比纯运算核心簇的性能略低；最好情况下，管理核心全部命中 Cache，则访主存带宽的总需求降低，性能提高的比例近似为管理核心承担任务量的比例。一般情况下，核组异构协同的性能略好于纯运算核心簇的性能。SW26010 处理器的每个核组搭配 1 个管理核心和 64 个运算核心，考虑到主核拥有 Cache，其任务量应略大于总任务量的 1/64。实验中采用静态任务划分，管理核心承担总任务的 1/32。另一方面，核组异构协同依赖 Cache 命中率意味着能够提高 Cache 命中率的方法都有利于核组异构协同 BFS 性能的提高。因此，我们对图做清洗和预处理，将邻接列表中指向自身的边剔除，然后将邻接列表中的有效数据压紧排齐，使不同顶点的邻接列表紧密排列，以提高数据局部性和 Cache 命中率，进而提高核组异构协同算法的性能。邻接列表压缩后数据量的占比如表 2 所列。

表 2 不同问题规模下压缩后邻接列表的数据量占比

Table 2 Percentage of data amount after compression under different problem sizes

Scale	邻接列表压缩后的数据量占比/%
18	90.75
20	93.58
22	95.60

4 实验结果

实验中，SW26010 的运行环境配置如表 3 所列。

表 3 SW26010 配置

Table 3 SW26010 configuration

结构	环境配置
管理核心	4 个
运算核心	4 个簇，每个簇 64 个运算核心
频率	1.45 GHz
SIMD 宽度	256 bit
管理核心存储体系	32 kB L1 数据 Cache 32 kB L1 指令 Cache 256 kB 数据指令共享 L2 Cache
运算核心存储体系	16 kB L1 指令 Cache 64 kB 私有数据存储，配置为 SPM
主存容量	8 GB x4
主存带宽	总带宽 27 GB/s，核组共享 管理核心的访存带宽最大为 9 GB/s， 运算核心簇可用满访存带宽

我们在 SW26010 上基于单核组实现了本文提到的优化，在上述环境配置下对不同 Graph 问题规模进行了测试，结果如图 9 所示。邻接列表按度数降序排列，可使 BFS 性能提高 15%～20%；主从协同，可使性能提高约 1%；将邻接列表中指向自身的边去除并压紧，可使 BFS 性能提高 0.1%～0.3%；将度数为 0 的点标记为 visited，可使性能提高约 2%。

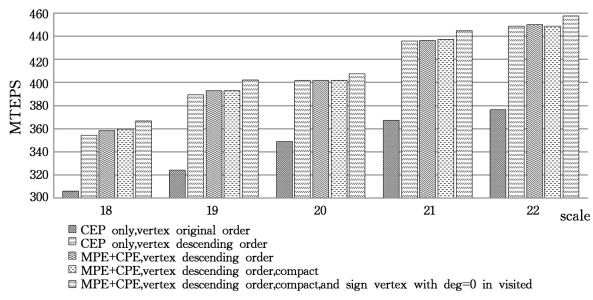

图 9 SW26010 上基于单核组不同规模、不同优化条件下的 BFS 测试结果

Fig. 9 BFS test results under differents scale with different optimizations on single CG of SW26010

最终，对于 *scale*=22 的图，SW26010 上单核组的 BFS 性能可达 457.54 MTEPS。

表 4 比较了本文工作与同时期 GPU 上实现的单机 BFS 的性能。当 BFS 单机并行时，算法瓶颈主要在访存带宽上，其性能应与访存带宽呈弱线性关系。表中 Ratio 列的值为将 BFS 性能的值进行访存带宽归一化处理后的结果。从表 4 可以看出，本文提出的方法优于大部分 GPU 上的实现。

表 4 近年 GPU 上单机 BFS 性能的比较

Table 4 Comparison of recent works implementing BFS on single GPU

Works	Year	Scale	MTEPS	Architecture	Single Precision Performance/Tflops	Band Width/(GB/s)	Ratio
Merrill[14]	2012	20	3100	1x Nvidia Tesla C2050	1.03	144	1.43
Fu[15]	2014	21	2500	1x Nvidia Tesla K20 GPU	3.52	208	0.73
Peter[16]	2015	24	725	1x Nvidia Tesla K40 GPU	4.29	288	0.15
Yang[17]	2018	21	63	1x Nvidia Tesla P100 GPU	9.3	732	0.01
Present Work	2019	20	407.43	1x SW26010 CG (A quarter of SW26010 CPU)	1.5312(整数运算性能)	27	1.00
		21	444.75				
		22	457.54				

结束语 本文基于 SW26010 异构众核处理器实现了单核组的方向优化融合 BFS 算法，结合体系结构的特点，使用异步 DMA 提高访存效率并隐藏计算和 SPM 访问开销；采用 tag-TD 和 byte-TD 优化 TD 算法，避免了锁的使用；将邻接列表按度数降序排列并将度数为 0 的点标记为 visited，提高了 BU 算法的效率；利用异构架构的特点，采用核组异构协同方式提高 BFS 性能，并通过压缩 CSR 格式的邻接列表提高协同算法的 Cache 命中率。未来基于 SW26010 的单核组 BFS，可以尝试重构图、负载平衡或动态任务划分等工作，更深入地利用簇内寄存器通信降低访主存带宽需求等。

参 考 文 献

[1] Graph 500[OL]. http://www.graph500.org.

[2] AGARWAL V, PETRINI F, PASETTO D, et al. Scalable Graph Exploration on Multicore Processors[C]// International Conference for High Performance Computing, Networking, Storage & Analysis. IEEE, 2010: 1-11.

[3] UENO K, SUZUMURA T. Highly scalable graph search for the Graph500 benchmark[C]// International Symposium on High-performance Parallel & Distributed Computing. 2012: 149-160.

[4] TITHI J J, MATANI D, MENGHANI G, et al. Avoiding Locks and Atomic Instructions in Shared-Memory Parallel BFS Using Optimistic Parallelization[C]// IEEE International Symposium on Parallel & Distributed Processing. IEEE, 2013: 1628-1637.

[5] CHHUGANI J, SATISH N, KIM C, et al. Fast and Efficient Graph Traversal Algorithm for CPUs: Maximizing Single-Node Efficiency[C]// IEEE International Parallel & Distributed Processing Symposium. 2012: 378-389.

[6] BEAMER S, BULUC A, ASANOVIC K, et al. Distributed Memory Breadth-First Search Revisited: Enabling Bottom-Up Search[C]// IEEE International Parallel & Distributed Processing Symposium Workshops & Phd Forum. 2013: 1618-1627.

[7] YASUI Y, FUJISAWA K, GOTO K. NUMA-optimized parallel breadth-first search on multicore single-node system[C]// IEEE International Conference on Big Data. IEEE, 2013: 394-402.

[8] YASUI Y, FUJISAWA K, SATO Y. Fast and Energy-efficient Breadth-First Search on a Single NUMA System[M]// Supercomputing. Cham: Springer, 2014: 365-381.

[9] TAO G, YUTONG L, GUANG S. Using MIC to Accelerate a Typical Data-Intensive Application: The Breadth-first Search [C]// IEEE International Symposium on Parallel & Distributed Processing. 2013: 1117-1125.

[10] BUSATO F, BOMBIERI N. BFS-4K: an Efficient Implementation of BFS for Kepler GPU Architectures[J]. IEEE Transactions on Parallel & Distributed Systems, 2014(1): 1-1.

[11] ZOU D, DOU Y, WANG Q, et al. Direction-Optimizing Breadth-First Search on CPU-GPU Heterogeneous Platforms[C]// IEEE International Conference on High Performance Computing & Communications & IEEE International Conference on Embedded & Ubiquitous Computing. IEEE, 2013: 1064-1069.

[12] CHECCONI F, PETRINI F, WILLCOCK J, et al. Breaking the speed and scalability Barriers for Graph exploration on distributed-memory machines[J]. International Conference for High Performance Computing, Networking, Storage & Analysis, 2012, 7196(5): 1-12.

[13] FU H, LIAO J, YANG J, et al. The Sunway TaihuLight supercomputer: system and applications[J]. Sciece China Information Sciences, 2016, 59: 1-16

[14] DUANE M, MICHAEL G, ANDREW G. Scalable GPU graph traversal[J]. Acm Sigplan Notices, 2012, 47(8).

[15] FU Z S, HARISH K D, BRADLEY B, et al. Parallel breadth first search on GPU clusters[C]// 2014 IEEE International Conference on Big Data(Big Data). IEEE, 2014.

[16] PETER Z, ERIC H, JOHN M, et al. Dynamic parallelism for simple and efficient GPU graph algorithms[C]// Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms. ACM, 2015.

[17] YANG H, HANG L, HUANG H H. High-performance triangle counting on gpus[C]// 2018 IEEE High Performance extreme Computing Conference(HPEC). IEEE, 2018.

YUAN Xin-hui, born in 1989, master, research associate. His main research interests include parallel algorithm design and optimization, MPI and benchmark optimization.