众核平台上广度优先搜索算法的优化

徐启泽 韩文廷 陈俊仕 安 虹

(中国科学技术大学计算机科学与技术学院 合肥 230027)

摘 要 图算法在多个领域具有重要的应用价值。随着社会信息化程度的提高,需要处理的图数据量越来越大,图算法的性能已成为研究热点。广度优先搜索算法是一种重要的图算法,研究它的性能优化技术可以为其他图算法的性能优化提供借鉴。目前,在新一代 Xeon Phi 众核处理器上的工作均基于自顶向下算法且没有考虑到非均匀访存(NUMA)对性能的影响。文中以混合广度优先搜索算法为基础,结合 NUMA 拓扑结构,从任务分配、向量化和数据预处理3个方面展开优化,在 Xeon Phi 平台上设计并实现了高性能并行广度优先搜索算法。一系列实验结果表明,优化后的算法在不同规模的测试数据上与 Graph500 官方优化的算法相比取得了 50%~145%的性能提升。

关键词 广度优先搜索,众核架构,非均匀访存,向量化,性能优化

中图法分类号 TP302 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.01.049

Optimization of Breadth-first Search Algorithm Based on Many-core Platform

XU Qi-ze HAN Wen-ting CHEN Jun-shi AN Hong

(School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)

Abstract Graph algorithms have important application value in many fields. With the development of social informatization, the amount of graph data to be processed has become larger and larger, the performance of graph algorithms has become a research hotspot. Breadth-first search algorithm is an important graph algorithm, studying its performance optimization techniques can provide reference for the performance optimization of other graph algorithms. The current work on new generation Xeon Phi many-core processors are based on top-down algorithm and do not take into account NUMA effects on performance. Based on the hybrid breadth-first search algorithm and the NUMA topology, this paper optimized Breadth-first search algorithm from the perspective of tasks allocation, vectorization and data preprocessing, designed and implemented a high-performance parallel breadth-first search algorithm on the Xeon Phi platform. A series of experimental results show that the optimized algorithm has gained 50% ~ 145% improvement compared with Graph500 official tuned algorithm on different scales of test data.

Keywords Breadth-first search, Many-core architecture, NUMA, Vectorization, Performance optimization

1 引言

图分析算法在社交网络分析、人工智能和计算机辅助设计等多个领域均具有重要应用价值,随着图数据量的增大,如何提高图分析算法的性能已成为研究热点。广度优先搜索(BFS)算法是一种重要的图搜索算法,它具有访存数据量大、访存模式不规则和访存局部性差的特点,很好地刻画了一类数据密集型应用的特征。研究 BFS 算法在众核处理器平台的优化技术对其他图算法和数据密集型应用在该平台上的优化有参考价值。作为一种典型的访存模式不规则程序,BFS 算法在运行过程中会产生大量的随机访存请求,Cache 和数据预取技术很难发挥较好的效果,访存速度是限制其性能的瓶颈。

本文以混合 BFS 算法为基础,针对 NUMA(非均匀访存)拓扑结构优化任务分配,结合向量化技术、数据预处理技术,在 Xeon Phi Knights Landing(KNL)众核处理器上设计并实现了高性能广度优先搜索算法。与官方的 OpenMP 实现相比,该方法的搜索速度提高了 50%~145%。

2 相关工作

在 NVIDIA GPU 上,Liu 等[1]从线程调度、负载均衡等方面进行优化,有效地提高了混合 BFS 算法[2]在 NVIDIA GPU 上的搜索性能。在神威太湖之光上,Lin 等设计并实现了适合超大规模异构系统的异步并行 BFS 算法,在超过一千万个处理器核上取得了 Graph500 世界排名第二的性能[3]。

在 Xeon Phi 众核处理器上,Paredes 等挖掘向量处理单

到稿日期:2018-01-18 返修日期:2018-04-05 本文受国家重点研发计划(2016YFB0201902)资助。

徐启泽(1992-),男,硕士生,主要研究方向为高性能计算;韩文廷(1965-),男,博士,副教授,主要研究方向为高性能计算;陈俊仕(1990-),男,博士生,主要研究方向为多核并行编程;安 虹(1963-),女,博士,教授,主要研究方向为超大规模并行计算机系统结构、大数据并行存储与处理系统、高性能计算,E-mail:han@ustc.edu.cn(通信作者)。

元的潜力，提出了一种向量化自顶向下 BFS 算法，使用向量处理单元并行搜索邻接点列表，使用一种恢复算法避免原子操作，与非向量化版相比，该方法获得了 1.3 倍加速比[4]。Gao 等[5]优化实现了混合 BFS 算法，与 CPU 版本相比获得了 1.4 倍加速比，该工作也采用了向量化技术，但该工作的图数据存放在空间较小的板载高速内存中，受空间限制只能处理较小的图。以上工作基于上一代 Xeon Phi 众核处理器。在新一代 Xeon Phi Knights Landing 众核处理器上，Besta 等提出了一种向量化表示格式 SlimSell，通过减少图数据的存储空间占用来降低访存压力，基于 SpMV（稀疏矩阵向量乘法）的 BFS 算法的性能与 Graph500 官方实现相比最多提高了 33%，但该工作没有考虑众核处理器上 NUMA 对访存性能的影响[6]。

在高性能计算领域，访存密集型程序针对 NUMA 拓扑结构进行优化是一种常见的优化方法。Agarwal 等最早在多核平台上针对 NUMA 拓扑结构对自顶向下 BFS 算法进行优化[7]。Yasui 等在多核平台上针对 NUMA 拓扑结构对混合 BFS 算法进行优化，通过图数据划分和细致的任务分配，使优化后的混合 BFS 算法与 Graph500 官方的自顶向下 BFS 实现相比取得了 12～124 倍的加速比[8]，但该工作没有研究向量化对 BFS 算法性能的提升。目前还没有在 Xeon Phi Knights Landing 众核处理器上针对 NUMA 拓扑结构对混合 BFS 算法进行优化的相关工作，本文的优化工作具有借鉴意义。

3　背景介绍

3.1　Graph500

Graph500 是美国 Sandia 国家实验室发布的基准测试程序，与 TOP500 采用的计算密集型基准测试程序 Linpack 不同，该程序通过在大规模 Kronecker 图上进行广度优先搜索并计算每秒遍历边数（TEPS）来衡量计算机系统的性能[9-10]。符合官方规范的 Graph500 测试有以下步骤：

1）图生成，使用 Kronecker 图生成器产生一系列如〈I，J〉的边，I 和 J 分别是边的两个端点。该过程以 $SCALE$ 和 $edgefactor$ 作为输入，输出一个顶点数为 2^{SCALE}、边数为 $2^{SCALE} * edgefactor$ 的 Kronecker 图。

2）图构建，以步骤 1）中生成的边作为输入，构建高效的图存储格式，如 CSR（压缩稀疏行）和 CSC（压缩稀疏列）[11]。

3）广度优先搜索，从图中随机选择 64 个顶点开始广度优先搜索并输出 BFS 生成树。

4）验证与结果输出，验证步骤 3）中所得 BFS 生成树的正确性，并计算每秒遍历边数作为性能评价依据。

为保证实验结果的科学性，本文采用 Graph500 基准测试程序的数据集、测试方法和性能评价标准。

3.2　Xeon Phi 众核处理器

Xeon Phi 众核处理器是 Intel 公司推出的一种面向高性能计算领域具有高度并行性的处理器产品。本文所使用的代号为 Knights Landing（KNL）的众核处理器是该系列的第二代产品。它拥有基于 Silvermont 微架构的处理器核，每个处理器核包含 2 个宽度为 512 bit 的向量处理单元，支持 AVX-512 指令集扩展和 4 路超线程技术（4-Ways SMT）[12]。

在本文使用的众核处理器上，两个处理器核构成一个 Tile，共享 1MB 二级缓存，没有三级缓存。片上 Tile 之间通过 Mesh 网络进行通信。片上 Tile 可以工作在 3 种不同的集群模式（Clustering Mode）下：All-To-All，Quadrant/Hemisphere 和 Sub-NUMA Clustering（SNC）-4/SNC-2[12]。在 All-To-All 模式下，内存地址在所有 Tag Directory（TD）中均匀分布。在 Quadrant 模式下，所有 Tile 被分为 4 个象限，每个象限中内存控制器与访问的地址空间映射到本象限的 TD 中。Hemisphere 模式与 Quadrant 相似，只是将所有 Tile 划分为两个象限。SNC 模式在 Quadrant 和 Hemisphere 模式的基础上，将处理器和内存对操作系统显示为 4 个或者 2 个 NUMA 结点。Quadrant/Hemisphere 和 SNC-4/SNC-2 模式具有较低的访存延迟。与前一代 Xeon Phi 不同，KNL 还配备了 MCDRAM（多通道 DRAM），其带宽达到 400GB/s 以上，可以工作在 Cache 模式下作为末级缓存（Last Level Cache，LLC）。

3.3　混合广度优先搜索算法

广度优先搜索（BFS）是一种常用的图搜索算法。输入由顶点集 V 及边集 E 构成的图 $G(V,E)$ 和搜索起点 s，BFS 算法首先访问 s，接着访问 s 的未访问邻接点 v_1，v_2，v_3，…，v_n，然后按照 v_1，v_2，v_3，…，v_n 的顺序访问每个顶点的未访问邻接点，直到图中所有与 s 有路径相通的顶点都被访问到，算法结束。

算法 1 中，搜索任务队列 $currQ$ 中包含了要在本层搜索的顶点，又称为活跃顶点。算法搜索活跃顶点的邻接点列表寻找未访问顶点，设置其访问状态为已访问并存入下一层搜索任务队列 $nextQ$。在一层搜索完成后交换 $currQ$ 和 $nextQ$，继续下一层搜索直到任务队列 $currQ$ 为空。数组 $visited$ 记录所有顶点的访问状态。数组 $tree$ 记录每个顶点在 BFS 生成树中的父顶点，即在搜索过程中的直接前驱。辅助函数 $ADJLIST(G,v)$ 用于从图 G 的存储结构中提取顶点 v 的邻接点列表。搜索任务队列 $currQ$ 中的顶点编号是随机的，且随着算法的执行动态改变。这种随机性和动态性是 BFS 算法中大量随机访存的根源。

算法 1　广度优先搜索算法

```
输入：G〈V，E〉，s
输出：tree
tree[v]=－1，∀v∈V //BFS 生成树初始化
visited={s}，∀v∈V //访问状态初始化
currQ={s} //搜索任务队列初始化
while currQ ≠ Ø do
  nextQ={}
    for all to_vis∈currQ do
      for all v ∈ ADJLIST(G，to_vis) do
        if v∉visited then//找到未访问顶点
          tree[v]=to_vis
          visited=visited ∪{v}
          nextQ=nextQ ∪{v}
        end if
      end for
```

```
    end for
  swap(nextQ,currQ)
endwhile
```

层同步并行广度优先搜索是一种并行 BFS 算法外层循环的算法[1]。在该算法中,图数据 G、访问状态数组 *visited* 和生成树 tree 被多个线程共享。线程在搜索过程中会搜索到已经被本层搜索中其他线程访问过的顶点,从而产生无效搜索。图 1 中,虚线表示一次无效搜索,因为它所指顶点已经被线程 1 访问过。这种经典的 BFS 算法从已访问顶点出发,寻找未访问顶点,被称为自顶向下(Top-down)搜索。这种算法在并行化后因重复发现而产生的无效搜索问题对搜索性能的影响很大。

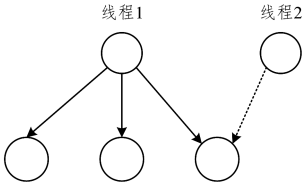

图 1 无效搜索示例

Fig. 1 Example of invalid search

采用自底向上(Bottom-up)搜索算法可以减少无效搜索[1]。与自顶向下搜索方向相反,自底向上搜索从未访问顶点 v 出发,搜索它的邻接点列表,寻找一个在上一层搜索中被访问且有出边与 v 相连的顶点 w,然后将 v 的前驱顶点设置为 w。与自顶向下搜索相比,自底向上搜索的显著优势在于:自底向上搜索中顶点只由一个线程确定在生成树中的父顶点,不存在重复发现现象。而且,搜索时只要在邻接点列表中找到一个在上一层搜索中被访问到的顶点即可结束搜索。这种快速结束特性可以在 BFS 算法的某些阶段减少无效的边搜索,降低访存次数。自底向上搜索算法的伪代码如算法 2 所示。

算法 2 自底向上搜索算法

```
输入:G〈V,E〉,visited,visited_in_last_level
输出:tree,visited
for v∈V\visited do //所有未访问顶点为活跃顶点
  for w∈ADJLIST(G,v) do
    if w∈visited_in_last_level//w 在上一层被访问
      tree[v] = w
      visited = visited ∪{v}
      break //结束搜索
    end if
  end for
end for
```

图 2 比较了两种不同搜索方法的边搜索次数。实验使用拥有 2^{20} 个顶点、$2^{20} * 16$ 条边的 Kronecker 图作为输入。从图 2 中可以看出,搜索的开始阶段自顶向下算法边搜索较少,中间阶段自底向上算法边搜索较少,快结束时自顶向下算法边搜索较少。Beamer 等根据这一现象提出了混合 BFS 算法,即将自顶向下搜索和自底向上搜索相结合的算法,在 BFS 的不同阶段使用不同的搜索算法,可以有效减少边搜索次数[2]。

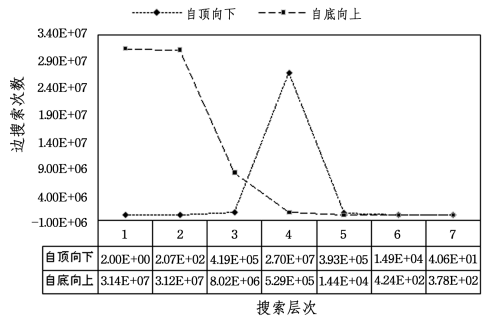

图 2 不同搜索算法边搜索次数的比较

Fig. 2 Comparison of edge search numbers of different search algorithms

这种混合 BFS 算法采用了一种启发式切换策略。在一层搜索的开始,算法估算在该层使用自顶向下搜索和自底向上搜索的边搜索次数,选择边搜索次数较少的搜索算法进行搜索。详细的切换策略请参见文献[2]。

4 优化方法

4.1 优化任务分配

非均匀访存(Non-Uniform Memory Access,NUMA)是一种针对多处理器计算机的内存访问设计[8]。在这种设计中,处理器和内存被组织成多个 NUMA 结点。与处理器处于同一结点的内存被称为该处理器的本地内存(Local Memory),而处于其他结点的内存被称为该处理器的远端内存(Remote Memory)。处理器访问本地内存比访问远端内存延迟低。广度优先搜索算法的性能在很大程度上取决于访存延迟,因此在采用 NUMA 设计的系统中优化 BFS 算法的搜索性能要考虑 NUMA 的特性,尽可能将访存限制在本地结点中以降低访存延迟。

当 Xeon Phi 众核处理器工作在 SNC-4 模式时,计算资源和存储资源被组织成 4 个 NUMA 结点。图 3 展示了实验使用的 Xeon Phi 众核处理器的 NUMA 拓扑结构。它的每个结点拥有 18 个处理器核(Core)和 24GB DDR4 内存。针对这种拓扑结构,本文使用一种可以避免访问远端内存的任务分配方式。其具体步骤如下:1)将图划分为多个子图,再将子图存储在不同的 NUMA 结点中;2)将线程与 NUMA 结点进行绑定,每个线程只搜索存储在本地内存的子图。

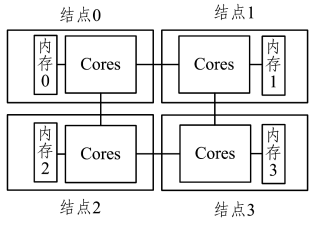

图 3 Xeon Phi 的 NUMA 拓扑结构

Fig. 3 NUMA topological structure of Xeon Phi

4.1.1 图划分

对于一个无向图 $G\langle V,N\rangle$ 和 N 个 NUMA 结点,将 G 按列进行一维划分[8,13]后得到 N 个子图:

$$\{G_0\langle V_0,E_0\rangle,G_1\langle V_1,E_1\rangle,\cdots,G_{N-1}\langle V_{N-1},E_{N-1}\rangle\} \quad (1)$$

$$V_k=\{v_i\in V\mid i\in[k*\frac{|V|}{N},(k+1)*\frac{|V|}{N}]\} \quad (2)$$

$$E_k=\{\langle v_i,v_j\rangle\mid v_i\in V,v_j\in ADJLIST(v_i)\bigcap V_k\} \quad (3)$$

图 4 展示了划分后图的邻接矩阵。NUMA 结点 k 中存储了 G_k，它包含所有顶点 V 到局部顶点 V_k 的出边和局部顶点 V_k 的所有入边。除图数据外，每个结点还存储了算法工作时的数据结构，结点 k 存储了以下数据结构：

$$visited_k=visited\bigcap V_k \quad (4)$$

$$currQ_k=currQ \quad (5)$$

$$nextQ_k=nextQ\bigcap V_k \quad (6)$$

$$tree_k=tree\bigcap V_k \quad (7)$$

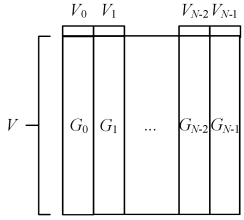

图 4　划分后图的邻接矩阵

Fig. 4　Adjucent matrix of partitioned graph

4.1.2　任务分配

对 *visited*，*currQ*，*nextQ* 和 *tree* 进行划分后，搜索任务随之被划分并分配到各 NUMA 结点。在自顶向下搜索阶段，结点 k 中的线程搜索 $currQ_k$ 中的活跃顶点的出边，将找到的未访问顶点加入 $nextQ_k$。在每层搜索的末尾，对所有 NUMA 结点中的线程进行同步，收集全部结点中的 $nextQ_i$（$0\leqslant i\leqslant N$）并合并成完整的 *nextQ*，再与 *currQ* 进行交换，进入下一层搜索。在自底向上搜索阶段，NUMA 结点 k 中的线程以 $visited_k$ 中标记为未访问的顶点作为活跃顶点，搜索它们的入边，寻找存在于 $currQ_k$ 中的顶点（$currQ_k$ 记录了在上一层被访问的顶点），并将本层搜索到的顶点加入 $nextQ_k$。同样地，在每层搜索的末尾，对每个结点中的线程进行同步和收集操作形成新的 *currQ*，再进入下一层搜索。

经过图划分和任务分配后，BFS 算法搜索时大部分访存都被限制在本地内存。收集操作引入的少量对远端内存的顺序访问速度很快，对性能的影响很小。除此之外，将图数据和运行时数据（*visited*，*currQ*，*nextQ*，*tree*）分割成多份存放在不同的节点后，结点的本地线程访问的内存区域变小，具有更好的访存局部性。对于缓存较小的硬件平台，这种分散的数据布局更有优势。

为了支持任务分配优化，Xeon Phi 工作在 SNC 模式，并通过 libnuma 库提供的绑定函数集合将线程绑定到特定的 NUMA 结点。

4.2　向量化自底向上搜索

Xeon Phi 众核处理器的每个核心拥有两个可以同时处理 512bit 数据的向量处理单元（Vector Processor Unit，VPU）。在自底向上搜索阶段，最耗时的操作是搜索每个未访问顶点的邻接点列表以寻找一个在上一层搜索被访问到的顶点。与 Paredes 使用的向量化自顶向下搜索不同，本文提出了一种向量化自底向上搜索方法，利用 VPU 并行自底向上算法的邻接点列表搜索操作。具体步骤如下：

1）为每个线程分配连续的 16 个顶点编号。使用_mm512_set_epi32()操作设置向量寄存器 vsearch。

2）利用 *visited* 中的信息，将 vsearch 中已经被访问的顶点设置为不活跃，不参与后面操作。

3）从图 G 中读取 vsearch 中未访问顶点的邻接点列表地址，并设置下标 vcheck_cnt 中的元素为 0。

4）以 vcheck_cnt 为访问邻接点列表的下标，使用_mm512_mask_i32gather_epi32()从 16 个邻接点列表中分别载入一个顶点到 vneighbor。检查 vneighbor 中在上一层被访问的顶点，如果找到这样的顶点，那么将 vsearch 中相应的顶点设置为已访问、不活跃，它的邻接点列表不参与下一次检查。使用_mm512_i32scatter_epi32()操作将结果写回相关数据结构。

5）将 vcheck_cnt 的下标加 1，如果某顶点的邻接点列表搜索完毕，则将其设置为不活跃。重复步骤 4）直到 vsearch 中所有顶点被访问或所有邻接点列表搜索完毕。

在上面步骤中，图的顶点使用 32 位整型表示，以_mm512 开头的函数是 Intel 编译器提供的用于操作向量处理单元的 API，以 v 开头的变量长度均为 512bit。如图 5 所示，使用 VPU 检查未访问顶点的邻接表，一次可以搜索 16 个顶点的邻接点列表。

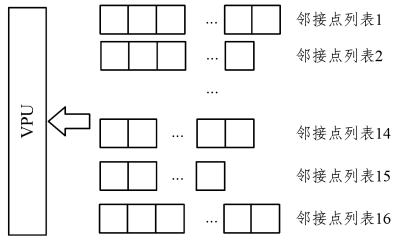

图 5　VPU 并行搜索邻接点列表

Fig. 5　VPU searchs adjacency list in parallel

向量化自底向上搜索算法的关键伪代码如算法 3 所示。

算法 3　向量化自底向上搜索算法

```
const vertex_t * inadj=g.get_inadj();
__m512i vsearch= _mm512_set1_epi32(-1);
__m512i vindegree=g.avx_getindegree(vs, vs_mask);
__m512i vcheck_cnt=g.avx_getinadjlist_start(vs, vs_mask);
__m512i vcheck_limit=__mm512_mask_add_epi32(_mm512_set1_
  epi32(-1), vs_mask, vcheck_cnt, vindegree);
do {
  __mmask16 not_found_mask =
  _mm512_cmp_epi32_mask(vsearch, _mm512_set1_epi32(-1),
  _MM_CMPINT_EQ);
  __mmask16 valid_mask =
  _mm512_cmp_epi32_mask(vcheck_cnt, vcheck_limit, _MM_
  CMPINT_LT);
  valid_mask =_mm512_kand(valid_mask, not_found_mask);
  if (_mm512_mask2int(valid_mask) == 0) break;
```

```
__m512i vneighbor=
_mm512_mask_i32gather_epi32(_mm512_set1_epi32(-1), valid_
mask,vcheck_cnt, inadj, sizeof(vertex_t));
__mmask16 last_vis_mask = past.avx_get(vneighbor, valid_
mask);
vsearch= _mm512_mask_mov_epi32(vsearch, last_vis_mask,
vneighbor);
vcheck_cnt=_mm512_add_epi32(vcheck_cnt, _mm512_set1_epi32
(1));
} while (1).
```

4.3 数据预处理

4.3.1 删除孤立顶点

输入图数据中会有一些顶点没有边与之相连，这种顶点被称为孤立顶点，不会被搜索到。对 Graph500 使用的 Kronecker 图进行分析后发现它含有大量孤立顶点，在一个拥有 2^{20} 个顶点、$2^{20}*16$ 条无向边的 Kronecker 图中共有 402004 个孤立顶点，占总顶点数的 38%。在自底向上阶段，使用 Bitmap 来表示搜索任务队列可以减小工作数据，提高缓存命中率。但 Bitmap 为所有顶点保留 1bit 空间，孤立顶点的存在使得 Bitmap 中出现大量无效位。为了提高 Bitmap 的利用效率，在保证正确性的情况下，对图中顶点进行重命名并删除所有的孤立顶点。

4.3.2 邻接点列表按度排序

在自底向上搜索阶段，只要找到一个在上一层被访问的顶点即可结束搜索。理想情况下，每个未访问顶点邻接点列表中第一个顶点即为上一层被访问到的顶点。然而，很难为所有顶点的邻接点列表找到一个最优顺序。本文对 Kronecker 图进行分析后发现图中顶点的度数并不是均匀分布的，少部分顶点的度非常高。考虑到度较高的顶点有更大概率在上一层被访问到，本文采用一种可以进一步减少边搜索次数的数据预处理方法，即在图的构建阶段，对图中顶点的邻接点列表按照度从高到低的顺序进行排序。由于邻接表中前面都是度较高的顶点，有更大的概率在这些顶点中找到一个在上一层被访问的顶点，避免了后续搜索。

在多核处理器上的相关工作采用了类似的预处理方法，并通过实验证明了以上两种预处理方法可以有效提高 BFS 算法的搜索性能[8]。

5 实验结果与分析

5.1 平台设置

实验平台基于 Intel Xeon Phi 7250F，主频为 1.4GHz，运行 CentOS 7.4 操作系统，具有 72 个处理器核(Core)，最多可支持 272 个线程。16GB 的 MCDRAM 工作在 Cache 模式作为末级缓存。所有的代码使用 C++编写，使用 OpenMP 作为多线程支持库，使用 Intel ICC 18.0 进行编译。实验采用 Graph500 的测试方法，对 Kronecker 图进行 64 次广度优先搜索并计算每秒遍历边数。在以下实验中，SnEm 代表顶点数为 2^n、边数为 2^n*m 的 Kronecker 图。所有测试使用 128 个线程。

5.2 性能测试结果

为比较任务分配优化和向量化带来的性能提升，选用顶点数为 $2^{20}\sim2^{23}$、*edgefactor* 为 16 的 Kronecker 图，测量优化任务分配和向量化前后的每秒遍历边数。从图 6 中可以看出，任务分配优化技术在测试数据集上可以获得 50%～120%的性能提升。

图 6 和图 7 分别给出了顶点数不同、edgefactor 相同和顶点数相同、*edgefactor* 不同的情况下向量化在优化任务分配基础上的性能对比。可以看出，在优化任务分配的基础上，向量化算法相对于非向量化算法可以获得 7%～10%的性能提升，且提升效果与图中边的密集程度有关，在顶点数相同的情况下，边越多(即 *edgefactor* 越大)性能提升幅度越大。

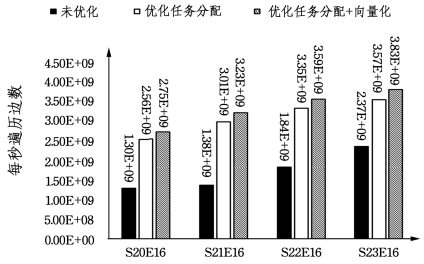

图 6 优化任务分配和向量化的性能提升

Fig. 6 Performance improvement of task allocation optimization and vectorization

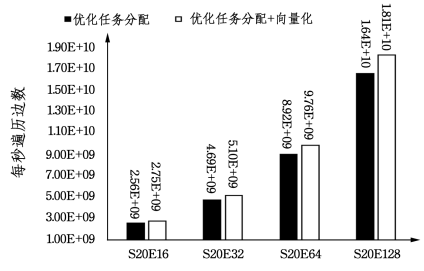

图 7 不同 edgefactor 图上向量化的性能提升

Fig. 7 Performance improvement of vectorization on graphs with different edgefactor

图 8 给出了在 *edgefactor* 为 16、顶点数不同的图上，在优化任务分配和向量化的基础上使用数据预处理优化前后的性能对比。在测试数据集上，数据预处理可以带来 10%～33%的性能提升。

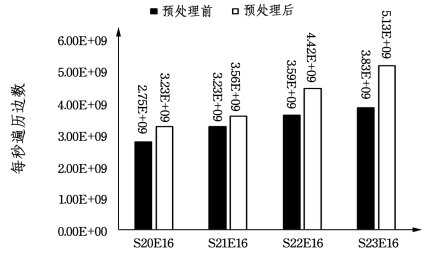

图 8 数据预处理的性能提升

Fig. 8 Performance improvement of data preprocessing

Graph500 官方提供了一个使用优化版混合 BFS 算法的实现 omp-csr-new[9]。本文实现与 omp-csr-new 在不同规模的 Kronecker 图上的性能比较，如图 9 所示。本文优化后算

法的每秒遍历边数与 omp-csr-new 相比提高了 50%～145%。

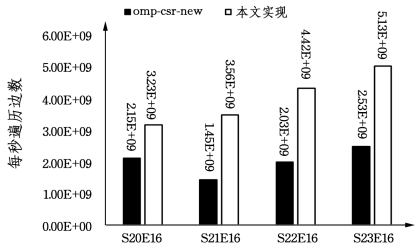

图 9　本文实现与 Graph500 官方实现 omp-csr-new 的对比

Fig. 9　Comparison between Graph500 official implementation omp-csr-new and realization in this paper

在 edgefactor 为 16，规模为 2^{19}，2^{20}，2^{21} 和 2^{22} 的Kronecker 图上，本文实现与 Paredes 等[4]在众核处理器上的实现相比有 2 倍以上的性能提升，与 Gao 等[5]在众核处理器上的优化实现相比最多快 24%。Besta 等[6]在 KNL 众核处理器上的优化实现比 Graph500 官方实现最多提高了 33%，低于本文实现的性能提升。

结束语　本文介绍了一种在 Xeon Phi Knights Landing 众核处理器上优化的并行广度优先搜索算法，实验结果表明，本文的算法与 Graph500 官方提供的优化版混合 BFS 算法相比有 50%～145%的性能提升。目前，自顶向下阶段仍有较多无效搜索且线程间负载不均衡问题较严重。下一步工作将以进一步减少混合 BFS 中自顶向下阶段无效搜索和改善各线程间的负载均衡为目标，提高 BFS 算法在 Xeon Phi 上的搜索性能。

参 考 文 献

[1] LIU H，HUANG H H. Enterprise：breadth-first graph traversal on GPUs[C]// SC15：International Conference for High Performance Computing，Networking，Storage and Analysis. 2015：1-12.

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

[3] LIN H，TANG X C，YU B W，et al. Scalable Graph Traversal on Sunway TaihuLight with Ten Million Cores[C]// International Parallel and Distributed Processing Symposium (IPDPS). 2017：635-645.

[4] PAREDES M，RILEY G，LUJAN M. Breadth first search vectorization on the Intel Xeon Phi[C]// Proceedings of the ACM International Conference on Computing Frontiers (CF'16). 2016：1-10.

[5] GAO T，LU Y T，ZHANG B D，et al. Using the Intel Many Integrated Core to accelerate graph traversal[J]. International Journal of High Performance Computing Applications，2014，28(3)：255-266

[6] BESTA M，MARENDING F，SOLOMONIK E，et al. SlimSell：A Vectorizable Graph Representation for Breadth-First Search [C]// 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS). 2017：32-41.

[7] AGARWAL V，PETRINI F，PASETTO D，et al. Scalable Graph Exploration on Multicore Processors[C]// In Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing，Networking，Storage and Analysis (SC'10). 2010：1-11.

[8] YASUI Y，FUJISAWA K. Fast and scalable NUMA-based thread parallel breadth-first search[C]// 2015 International Conference on High Performance Computing & Simulation (HPCS). 2015：377-385.

[9] Graph500[OL]. https：//graph500. org.

[10] LESKOVEC J，CHAKRABARTI D，KLEINBERG J，et al. Kronecker Graphs：An Approach to Modeling Networks[J]. The Journal of Machine Learning Research，2010，11(3)：985-1042.

[11] Sparse Matrix[OL]. https：//en. wikipedia. org/wiki/Sparse_matrix.

[12] SODANI A. Knights landing (KNL)：2nd Generation Intel® Xeon Phi processor[C]// 2015 IEEE Hot Chips 27 Symposium (HCS). 2015：1-24.

[13] HENG D D，TANG Y H，YI X D，et al. Implementation and performance analysis of BFS algorithm on a parallel prototype system[J]. Computer Engineering & Science，2017，39(1)：27-34. (in Chinese)
衡冬冬，唐玉华，易晓东，等. 并行原型系统上 BFS 算法设计实现与测试分析[J]. 计算机工程与科学，2017，39(1)：27-34.