Subway: Minimizing Data Transfer during Out-of-GPU-Memory Graph Processing

作者： Amir Hossein Nodehi Sabet， Zhijia Zhao， Jasmina Rajiv Gupta

University of California, Riverside

期刊： EuroSys '20

Abstract

提出了一种图计算系统Subway，是一种**out-of-GPU图处理系统**

主要贡献创新

- 设计了高效的子图生成技术，可以的减小每一次迭代加载数据
- 提出了异步子图处理

Introduction

- **研究动机**：

  在图处理系统中，数据传输的时间占主要的时间开销

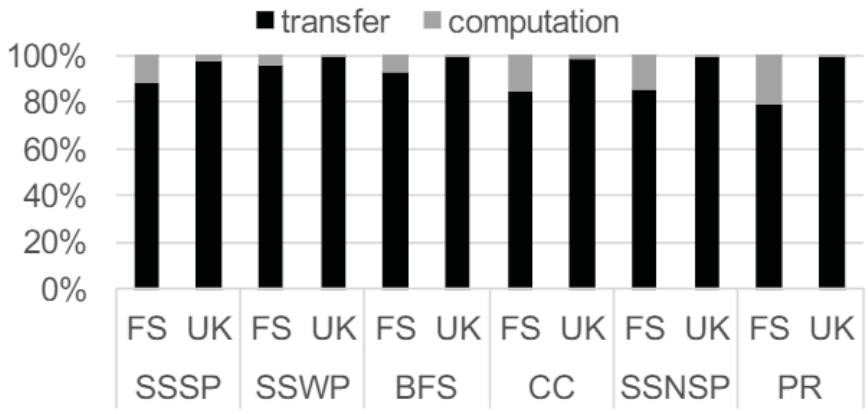


Figure 1. Time Breakdown of Partitioning-based Approach (six analytics on two real-world graphs from Section 4).

每一次加载的数据，有很多在近期计算中不会使用的点集/边集

Table 1. Ratio of Active Vertices and Edges

V-avg (max): average ratio of active vertices (maximum ratio);
E-avg (max): average ratio of active edges (maximum ratio).

Algo.	friendster-snap [6]		uk-2007 [4]	
	V-avg (max)	E-avg (max)	V-avg (max)	E-avg (max)
SSSP	4.4% (43.3%)	9.1% (85.1%)	4.6% (60.4%)	5.1% (67.7%)
SSWP	2.1% (38.4%)	5.2% (78.3%)	0.6% (12.6%)	0.6% (12.4%)
BFS	2.1% (32.3%)	4.1% (75.8%)	0.6% (12.6%)	0.6% (12.4%)
CC	8.1% (100%)	9.8% (100%)	3.2% (100%)	3.2% (100%)
SSNSP	2.1% (32.3%)	4.1% (75.8%)	0.6% (12.6%)	0.6% (12.4%)
PR	6.6% (100%)	24.1% (100%)	1.1% (100%)	1.7% (100%)

系统设计

- **高效的子图生成**

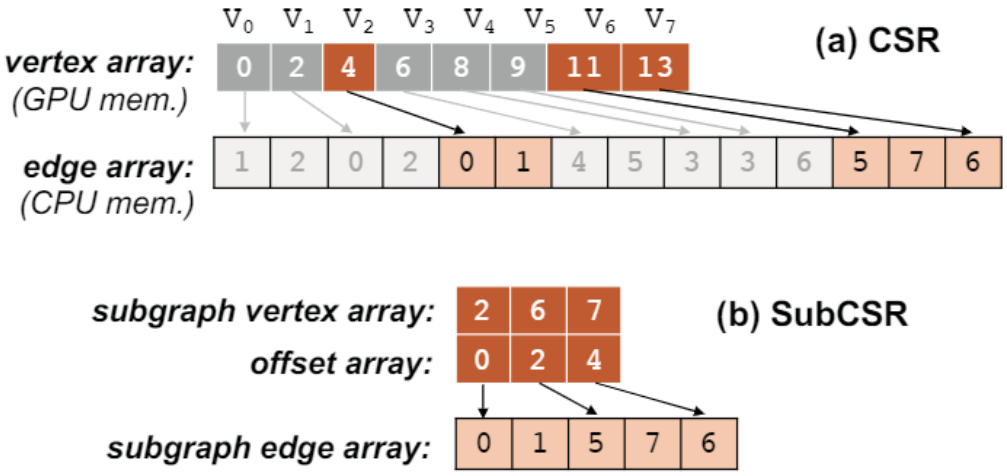


Figure 3. SubCSR Representation.

子图的表示：subgraph vertex array：原始点序列活跃点的位置、offset array：边在子图边序列的起始位置、subgraph edge array

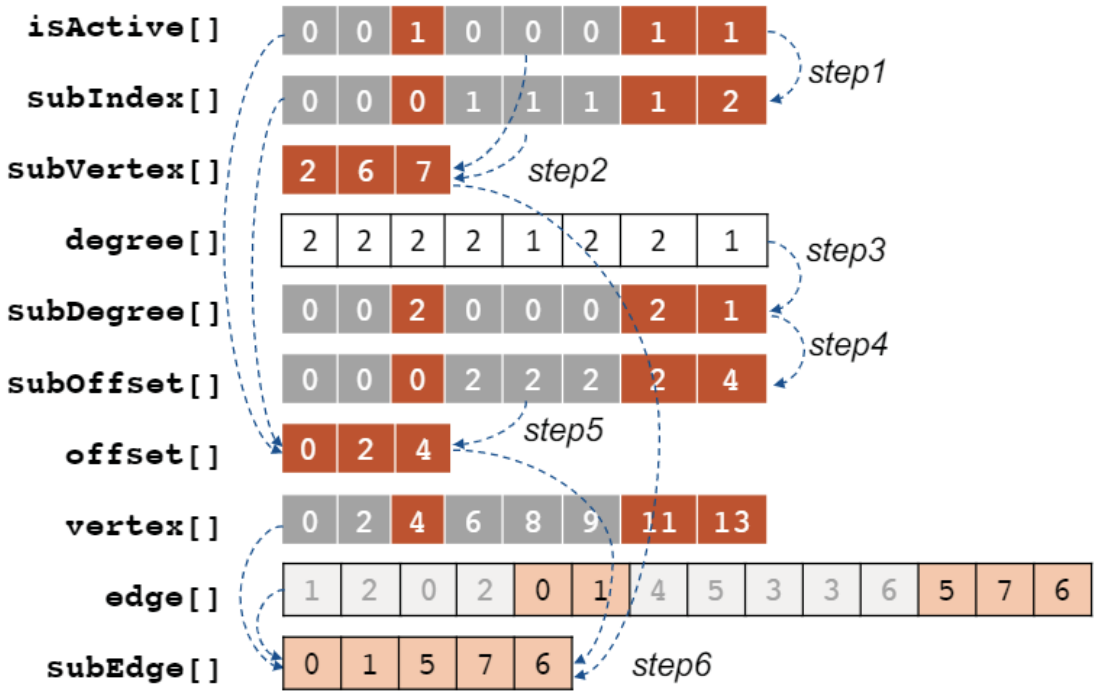


Figure 4. SubCSR Generation for Example in Figure 3.

生成子图表示SubCSR的过程

时间复杂度 $O(|V| + |E_{active}|)$

经过计算在active的点达到80%以下，使用这种子图生成的效率更高

- **异步图处理**

  传统两种方式计算点函数：

  - 同步方法
    - 在每次迭代结束后同步（BSP）
  - 异步方法
    - 在迭代内异步

  两种方式在每个迭代仅计算一次顶点。 这简化了图算法的开发，但导致数据传输率计算不足，从而使数据传输成为严重瓶颈。 为了克服这一障碍，Subway提供了更灵活的顶点函数计算策略-异步子图处理

  思想来源：**TLAG**：为了减少在分布式系统中生成的消息量，TLAG将编程抽象从顶点级别提升到分区级别。 在新的抽象中，图形编程变成了分区感知的，允许顶点值在每个分区内自由传播。由于顶点值传播到其他分区时可能变得“更稳定”，因此跨分区的通信趋于减少，因此减少到分布式系统中生成的消息。

Algorithm 4 Asynchronous Subgraph Processing

```
1:  do /* whole-graph-level iteration */
2:      V_active = getActiveVertices(G)
3:      G_sub = genSubCSR(G, V_active)
4:      /* the subgraph may be oversized, thus partitioned */
5:      for P_i in partitions of subgraph G_sub do
6:          load P_i to GPU memory
7:          do /* partition-of-subgraph-level iteration */
8:              for v_i in vertices of P_i do
9:                  f(v_i) /* evaluate vertex function */
10:             end for
11:         while anyActiveVertices(P_i) == 1
12:     end for
13: while anyActiveVertices(G) == 1
```

Iter.	V_0	V_1	V_2	V_3	V_4	V_5	V_6	V_7
0	0	1	2	3	4	5	6	7

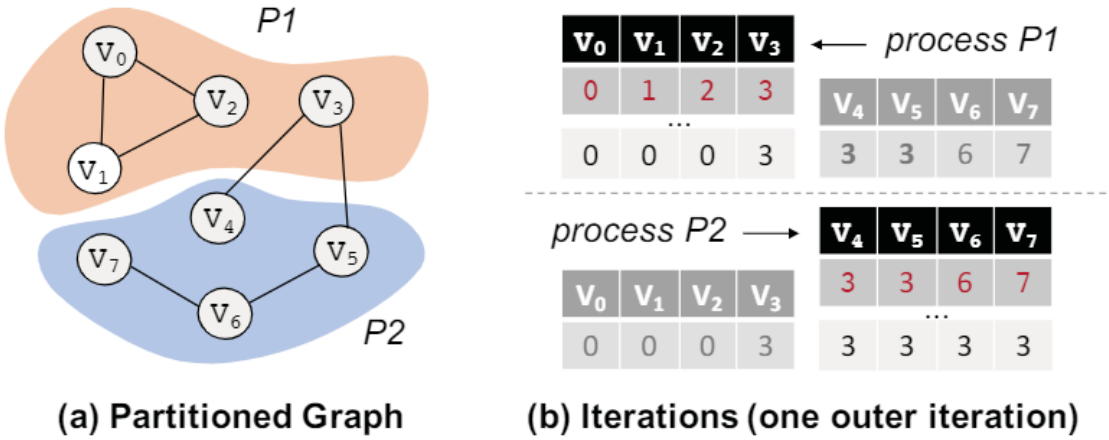


Figure 5. Example under Asynchronous Model.

该图为一个例子：P1和P2进行计算，首先P1计算结束后V4=V5=3，在P2计算时候可以使用到这个新值

正确性：异步处理会改变顶点传播的优先级，在一些对值传播顺序敏感的算法上可能会出现结果不一致的情况。

Experiments

环境

12 GB内存的NVIDIA Titan XP GPU

128 GB RAM的64核Intel Xeon Phi 7210处理器组成。

Datasets

Table 2. Graph Datasets

|V|: number of vertices; |E|: number of edges; Est. Dia.: estimated diameter range; $Size_w$: in-memory graph size (CSR) with edge weights; $Size_{nw}$: in-memory graph size (CSR) without edge weights.

Abbr.	Dataset	|V|	|E|	Est. Dia.	$Size_w$	$Size_{nw}$
SK	sk-2005 [4]	51M	1.95B	22-44	16GB	8GB
TW	twitter-mpi [3]	53M	1.96B	14-28	16GB	8GB
FK	friendster-konect [3]	68M	2.59B	21-42	21GB	11GB
FS	friendster-snap [6]	125M	3.61B	24-48	29GB	15GB
UK	uk-2007 [4]	110M	3.94B	133-266	32GB	16GB
RM	RMAT [11]	100M	10.0B	5-10	81GB	41GB

Methodology

PT(Basic Partitioning-based Approach):基于基本分区的方法

PT-Opt(Optimized Partitioning-based Approach):基于分区的优化方法

(UM-Opt)Optimized Unified Memory-based Approach:优化的基于统一内存的方法

效果

-

Table 3. Performance Results

Numbers (speedups) in bold text are the highest among the five methods;
Numbers (speedups) in italics are actually slowdown comparing to PT.

		PT	PT-Opt	UM-Opt	Subway-sync	Subway-async
SSSP	SK	118.3s	1.5X	3.5X	5.8X	**9.5X**
	TW	20.4s	1.7X	*0.8X*	2.9X	**6.0X**
	FK	53.0s	1.7X	*0.6X*	4.2X	**8.0X**
	FS	68.5s	1.6X	*0.7X*	4.2X	**6.7X**
	UK	492.7s	2.9X	1.9X	6.5X	**15.6X**
	RM	66.6s	1.3X	*0.6X*	2.0X	**3.1X**
SSWP	SK	174.7s	1.8X	5.2X	13.2X	**23.1X**
	TW	19.7s	2.2X	1.2X	4.4X	**7.0X**
	FK	50.3s	2.1X	1.2X	7.4X	**13.1X**
	FS	71.3s	1.8X	1.1X	8.0X	**12.5X**
	UK	350.8s	3.7X	4.9X	**38.8X**	36.3X
	RM	58.3s	1.1X	*0.5X*	2.2X	**3.7X**
BFS	FS	30.9s	1.9X	*0.9X*	6.7X	**9.6X**
	UK	176.3s	3.2X	10.3X	**28.8X**	21.8X
	RM	32.7s	1.5X	*0.7X*	3.2X	**3.7X**
CC	FS	22.9s	2.1X	1.1X	4.2X	**5.7X**
	UK	388.1s	5.7X	4.0X	10.9X	**26.0X**
	RM	25.5s	1.3X	*0.5X*	1.9X	**2.3X**
SSNSP	FS	59.1s	1.5X	*0.9X*	4.4X	**5.6X**
	UK	349.4s	4.0X	8.9X	**25.6X**	25.2X
	RM	61.7s	1.1X	*0.8X*	**4.6X**	3.9X
PR	FS	278.4s	2.5X	1.9X	**2.8X**	2.2X
	UK	577.9s	2.7X	3.4X	16.5X	**41.6X**
	RM	319.5s	1.2X	*0.9X*	**3.2X**	3.0X
GEOMEAN			2.0X	1.5X	6.0X	**8.5X**

•

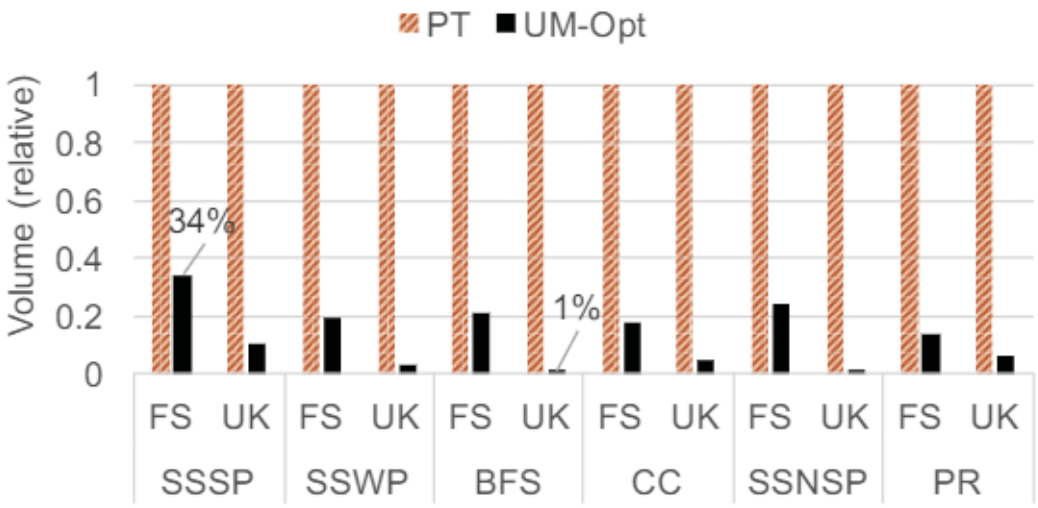


Figure 6. CPU-GPU Data Transfer (by volume).

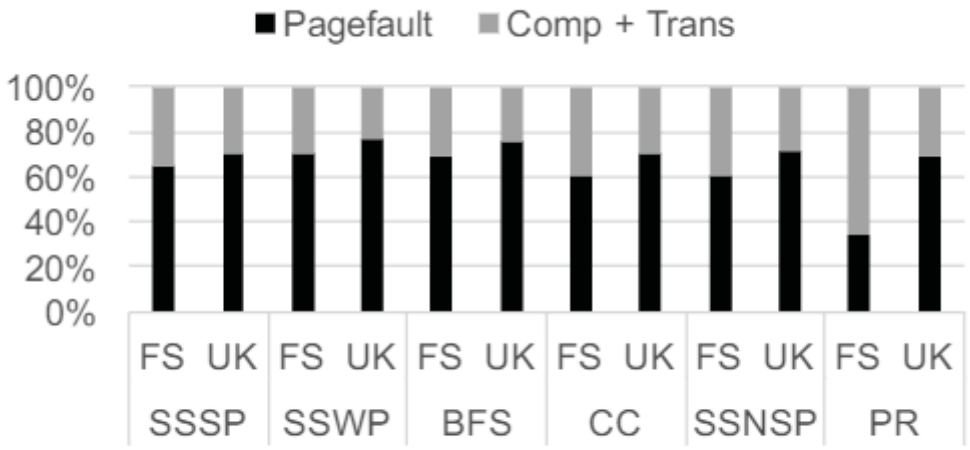


Figure 7. Page Fault Overhead in Unified Memory.

可以看到使用统一内存的方法，对于减少数据传输有着明显的效果，但是在实验中还是达不到稳定的效率，是因为其因为页错误开销的时间太多。

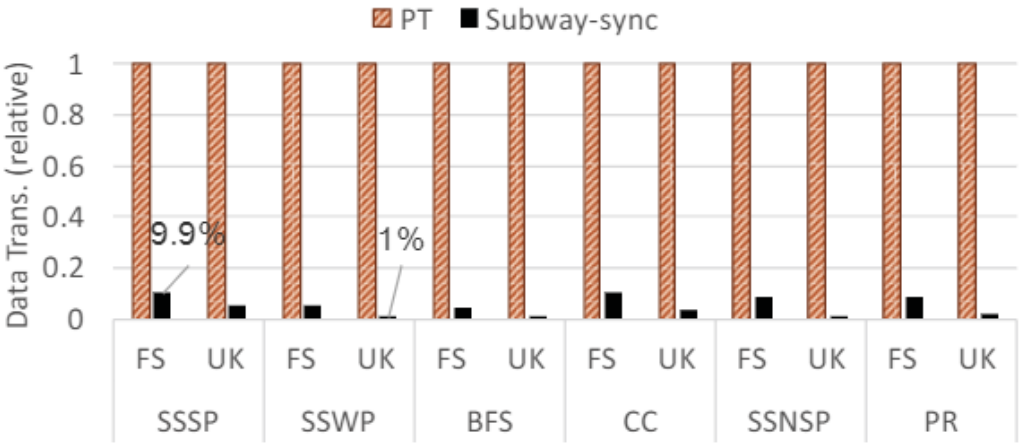
- 

Figure 8. CPU-GPU Data Transfer (by volume).

-

Table 5. Subway (Out-of-GPU-memory) vs. Galois (CPU)

GPU: Titan XP (3840 cores, 12GB);
CPU: Xeon Phi 7210 (64 cores); RAM: 128GB

		Subway-sync	Subway-async	Galois
SSSP	SK	20.28s	12.51s	**5.42s**
	TW	6.94s	**3.41s**	7.94s
	FK	12.54s	**6.65s**	22.129s
	FS	16.17s	**10.26s**	29.28s
	UK	75.47s	31.54s	**13.44s**
	RM	33.1s	**21.49s**	29.63s
BFS	FS	4.65s	**3.21s**	8.35s
	UK	6.12s	8.1s	**5.07s**
	RM	10.2s	**8.72s**	17.32s
CC	FS	5.49s	**4.03s**	5.23s
	UK	35.76s	14.93s	**4.88s**
	RM	13.7s	11.11s	**10.47s**

并且和目前最好的基于CPU的Galois图处理系统进行对比，发现性能还是可以媲美。

（作为GPU内存之外的解决方案，Subway的时间不仅包括从CPU到GPU的所有数据传输时间，还包括SubCSR生成时间。）

Conclusion and Future Works

对于基于GPU的图形处理，要解决的问题是超过GPU大小的数据是具有挑战性的问题。

论文提取仅由活动顶点的边组成的子图提供了极具成本效益的解决方案。大大减少了CPU和GPU之间的数据传输量。 减少数据传输的好处超过了图处理的几乎所有迭代中子图生成的成本，带来了实质性的整体性能提升。

论文还了用于GPU内存子图处理的异步模型，该模型可以安全地应用于广泛的图形算法，进一步提高了处理效率。

在实验中，Subway不仅与现有技术进行了比较，而且还与基于统一内存的优化实现进行了比较。 结果证实了有效性和效率。