Practical Optimal Cache Replacement for Graphs

作者：Vignesh Balaji, Neal Crago, Aamer Jaleel, Brandon Lucia

Carnegie Mellon University, Nvidia Research

期刊： HPCA 2021 Preprint

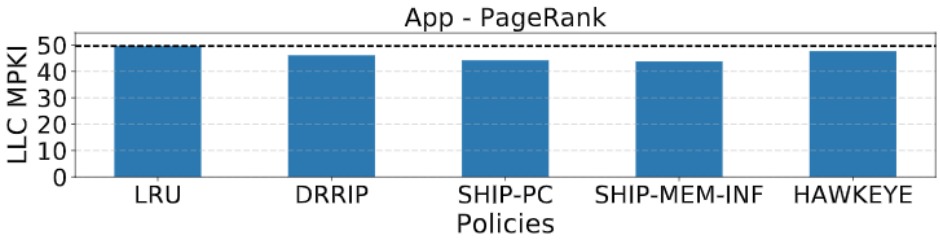
Abstract

由于不好的cache局部性，图分析一直不能达到最佳的效率

本文提出P-OPT的方法，使用转置下一次引用信息的压缩表示去有效的模拟最有的替换策略。

效率平均提高了**22%**。

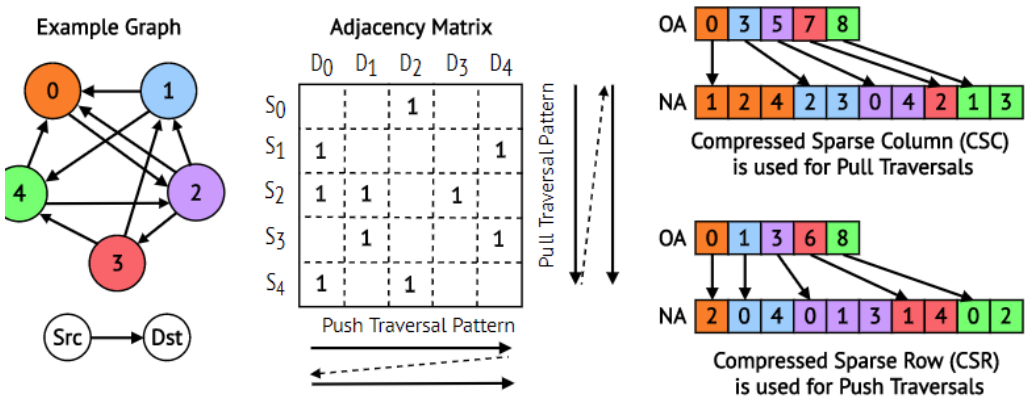
Introduction

- **研究动机**：
  - 图内核计算会花80%的时间在等待DRAM上面
  - 现有的替换策略不适合图结构处理
    - 图形数据的重用是动态可变的，并且与图形结构有关
    - 原先的替换策略假设一条指令的所有访问都具有相同的重用指令，但是图上度数不同的节点重用概率是不同的
    - 

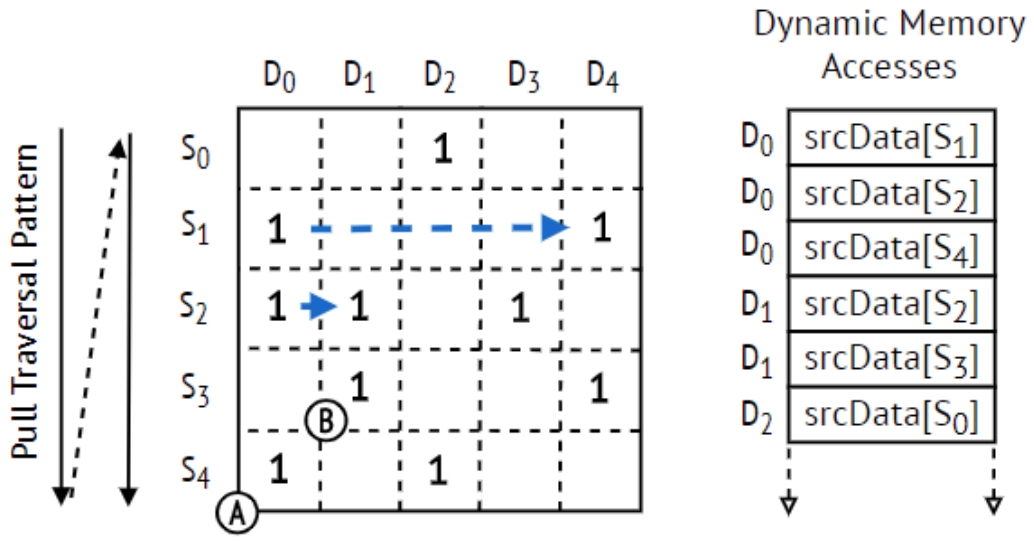
- **图处理过程**
  - **Algorithm 1** Pull execution of a graph kernel

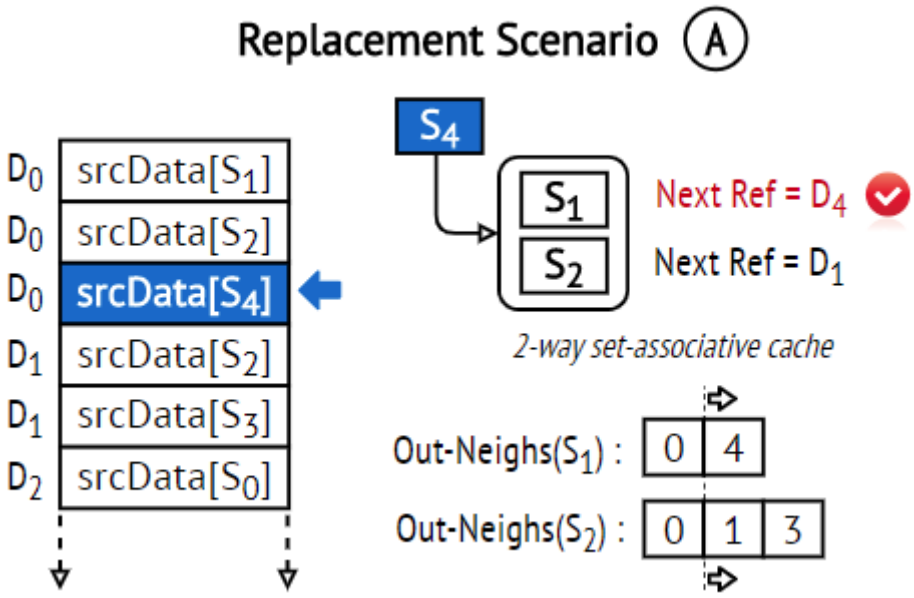
    ```
    for dst in G do
        for src in G.in_neighs(dst) do
            dstData[dst] += srcData[src]
    ```

    在访问邻接节点是随机的
- **CSC & CSR** （图压缩表示）
  -

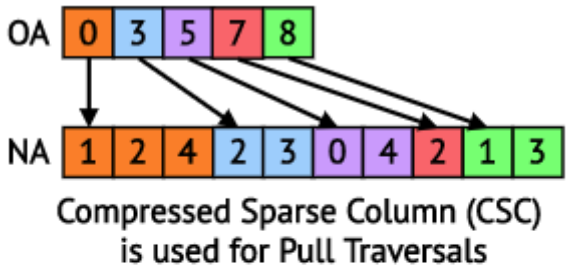


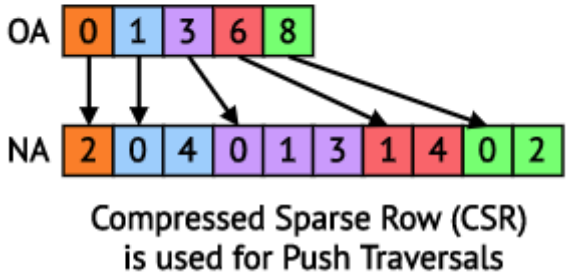
CSC就是按列压缩，OA数组代表终点D，值为其连接的起点个数，NA就是保存对应终点的起点。CSR同理

- **T-OPT** （一个理想的方法）
  - 

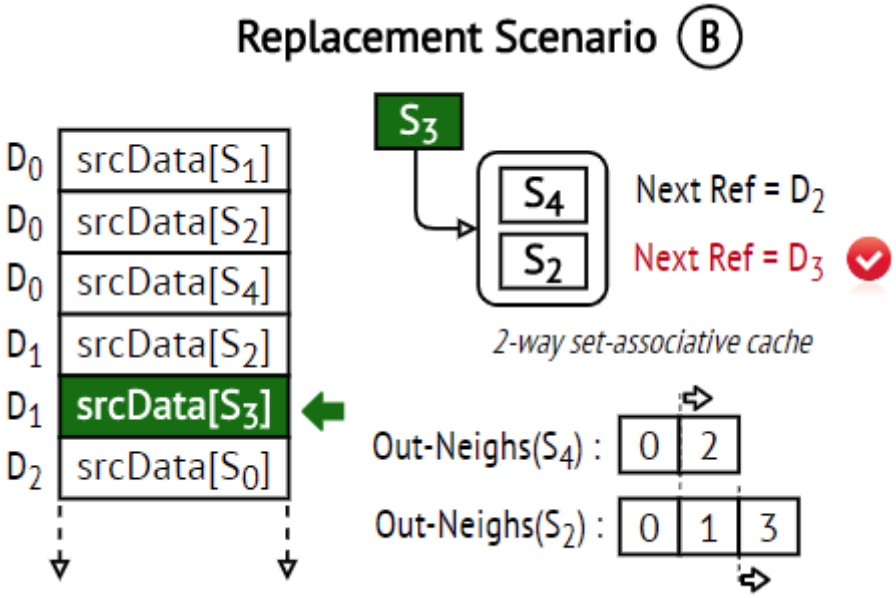
  - 假设我们使用前面的算法Pull，那么我们访问的srcdata就如右边所示
  - 

  - 假设一行cache只能放一个data，一共两行，那么在访问 S_4 的时候，就需要替换，那么更具OPT，替换未来最晚使用的就是 S_1 ，那么我们可以通过CSR来找到每一个数据的下一次访问是什么时候。





- 下次替换也是同理

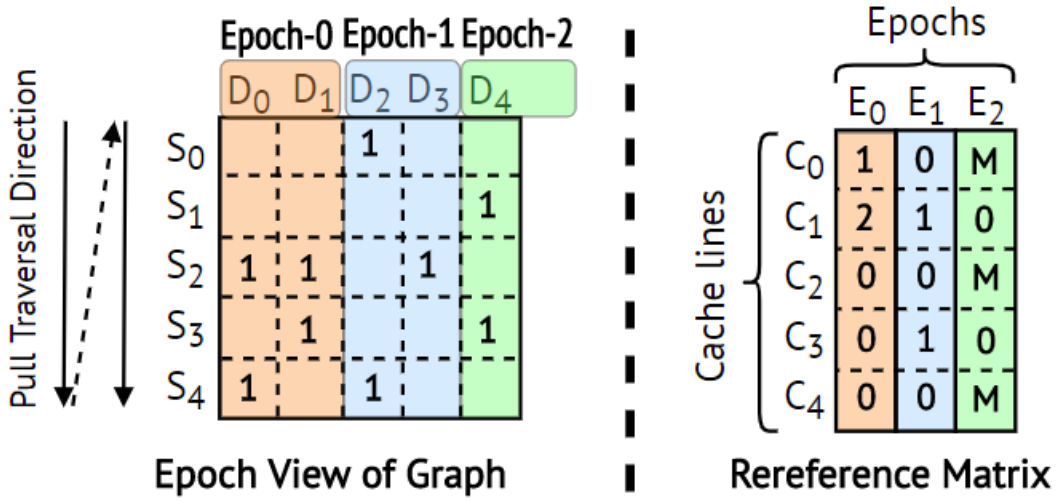


- **T-OPT的局限性**
  - 运行时间开销
    - 对于一个点，查找需要 $O(|d|)$ 的复杂度，如果cache一行有多个的话，就会费时很久
  - 缓存空间开销
    - 计算每行的下一个引用需要访问CSR。，由于驻留在缓存中的顶点可以是任意的，会导致额外的不规则内存访问，从而增加了图应用程序数据对缓存的争用。

P-OPT

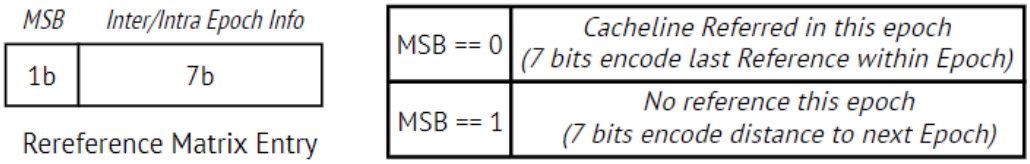
对T-OPT缺点的改进

-



首先对于列分epoch，假设每个epoch是2个顶点，对于行直接变成cache line，里面就是每一个cache line存的个数，因为每次换一行，其中的数值就不是是否有边了，而是下一次引用还有几个epoch。

- 这样还是会出现误差，特别是在epoch内存在引用的时候，再次改进



对于里面存在8bit，可以将第一位来表示是否有引用在该epoch，如果有的话，就用后7位表示最后一个引用的位置。

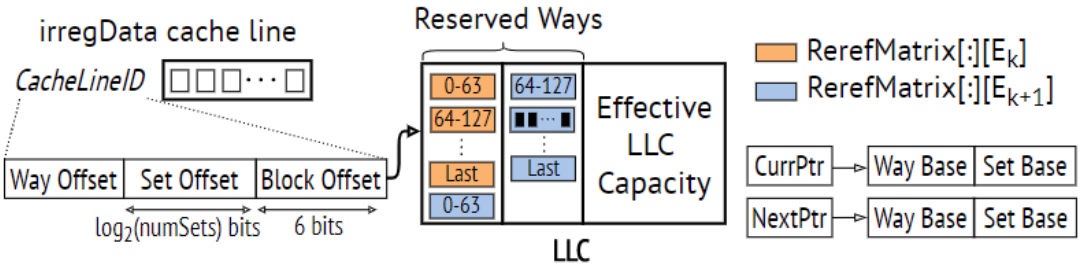
算法如下：

Algorithm 2 Finding the next reference via Rereference Matrix

```
1:  procedure FINDNEXTREF(clineID, currDstID)
2:      epochID ← currDstID/epochSize
3:      currEntry ← rerefMatrix[clineID][epochID]
4:      nextEntry ← rerefMatrix[clineID][epochID + 1]
5:      if currEntry[7] == 1 then
6:          return currEntry[6 : 0]
7:      else
8:          lastSubEpoch ← currEntry[6 : 0]
9:          epochStart ← epochID * epochSize
10:         epochOffset ← currDstID − epochStart
11:         currSubEpoch ← epochOffset/subEpochSize
12:         if currSubEpoch ≤ lastSubEpoch then
13:             return 0
14:         else
15:             if nextEntry[7] == 1 then
16:                 return 1 + nextEntry[6:0]
17:             else
18:                 return 1
```

P-OPT architecture

-



将需要的当前节点和下一个节点的引用矩阵存在LLC中，使用两个Ways，这两个Ways是保留的不会被替换。使用两个寄存器指向当前节点引用和下一个节点引用。

对于不规则访问数据流标记其在cache中位置，可以通过tag判断是否为不规则访问数据

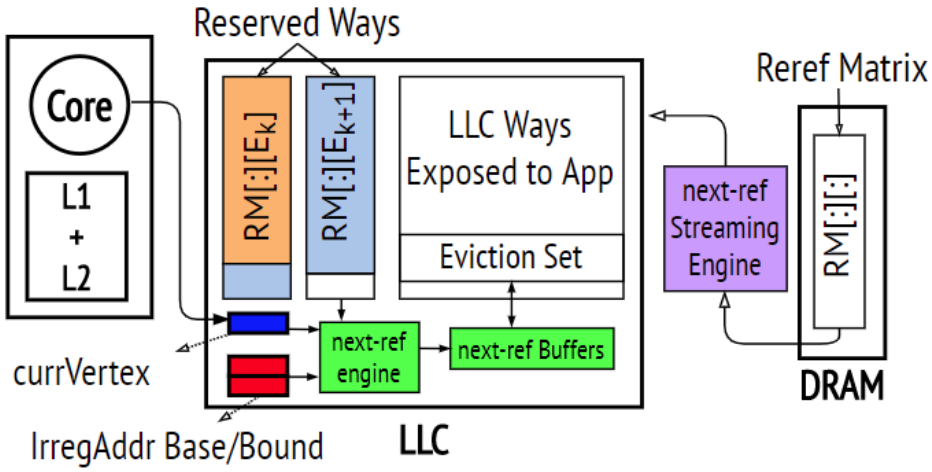
- 

Fig. 9: **Architecture extensions required for P-OPT:** *Components added to a baseline architecture are shown in color.*

底层设计图

- P-OPT扩展
  - 多核NUCA
  - 并行执行
  - 多不规则流数据
  - 虚拟化

复杂度： 只有 $next-ref\ \ buffers$ 需要额外的空间，但是不是很大一般1.25KB足够，$next-ref\ \ engine$ 是一个简单的FSM，只需要位运算和除法

Experiments

Parameters

Cores	8 OoO-cores, 2.266GHz, 4-wide issue, 128-entry ROB, Pentium M BP
L1(D/I)	32KB, 8-way set associative, Bit-PLRU replacement, Load-to-use = 3 cycles
L2	256KB, 8-way set associative, Bit-PLRU replacement, Load-to-use = 8 cycles
LLC	3MB/core, 16-way set associative, DRRIP replacement [24], Load-to-use = 21 cycles (local NUCA bank), NUCA bank cycle time = 7 cycles
NoC	Ring interconnect, 2 cycles hop-latency, 64 bits/cycle per-direction link B/W, MESI coherence
DRAM	173ns base access latency

TABLE I: **Simulation parameters**

Datasets

	DBP	UK-02	KRON	URAND	HBUBL
# Vertices (in M)	18.27	18.52	33.55M	33.55M	21.20
# Edges (in M)	136.53	292.24	133.51	134.22	63.58

TABLE III: **Input Graphs**: *All graphs exceed the LLC size*

Applications

	PR [6]	CC [6]	PR-δ [43]	Radii [43]	MIS [43]
irregData Elem Size	4B	4B	8B & 1bit	8B & 1bit	4B & 1bit
Execution style	Pull-Only	Push-Only	Pull-Mostly	Pull-Mostly	Pull-Mostly
Transpose	CSR	CSC	CSR	CSR	CSR
Uses frontier	N	N	Y	Y	Y

TABLE II: **Applications**

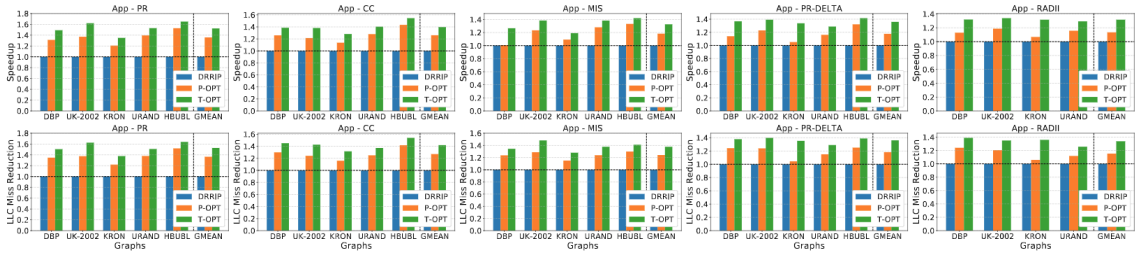


Fig. 10: **Speedups and LLC miss reductions with P-OPT and T-OPT**

Conclusion and Future Works

提出和介绍了P-OPT。

这是对于OPT替换策略的有效实施算法。

P-OPT利用了图形的转置编码了下一个引用来实现增强的缓存局部性和性能改进。

通过与图形无关的优化，P-OPT可提供更好的效果