李世阳，彭钰婷

研究生、本科生

南开大学

基于 oneAPI 的大规模图计算异构加速框架设计

- 图计算广泛应用于交通、社交、生物信息网络分析等真实场景，而 GPU 是常用的图计算加速工具。我们发现在图计算过程中，应用程序总是在一个很大的访存范围内对图数据进行顺序扫描;
- 通过划分GPU显存来分别存储可复用数据和其他数据，以优化程序的访存行为;
- 使用了 Intel oneAPI 作为编程工具实现了该框架的原型 OneGraph，它的内存管理方式能够有效减少 GPU 与 CPU 之间的数据传输, 并通过异步传输掩盖其延迟，提升图计算应用在 GPU 上的性能

南开大学嵌入式系统与信息安全研究室在读硕士研究生，主要研究方向为高性能计算、操作系统。

南开大学计算机科学与技术专业在读本科生，曾经参与高性能计算和计算机视觉有关工作，目前研究兴趣是计算机视觉。

李世阳，彭钰婷

研究生、本科生

南开大学

基于 oneAPI 的大规模图计算异构加速框架设计

- 图计算广泛应用于交通、社交、生物信息网络分析等真实场景，而 GPU 是常用的图计算加速工具。我们发现在图计算过程中，应用程序总是在一个很大的访存范围内对图数据进行顺序扫描;
- 通过划分 GPU 显存来分别存储可复用数据和其他数据，以优化程序的访存行为;
- 使用了 Intel oneAPI 作为编程工具实现了该框架的原型 OneGraph，它的内存管理方式能够有效减少 GPU 与 CPU 之间的数据传输，并通过异步传输掩盖其延迟，提升图计算应用在 GPU 上的性能

南开大学嵌入式系统与信息安全研究室在读硕士研究生，主要研究方向为高性能计算、操作系统。

南开大学计算机科学与技术专业在读本科生，曾经参与高性能计算和计算机视觉有关工作，目前研究兴趣是计算机视觉。

南開大學
Nankai University

OneGraph: A Cross-Architecture Framework for Large-Scale Graph Computing on GPUs Based on oneAPI

汇报人: 李世阳，彭钰婷

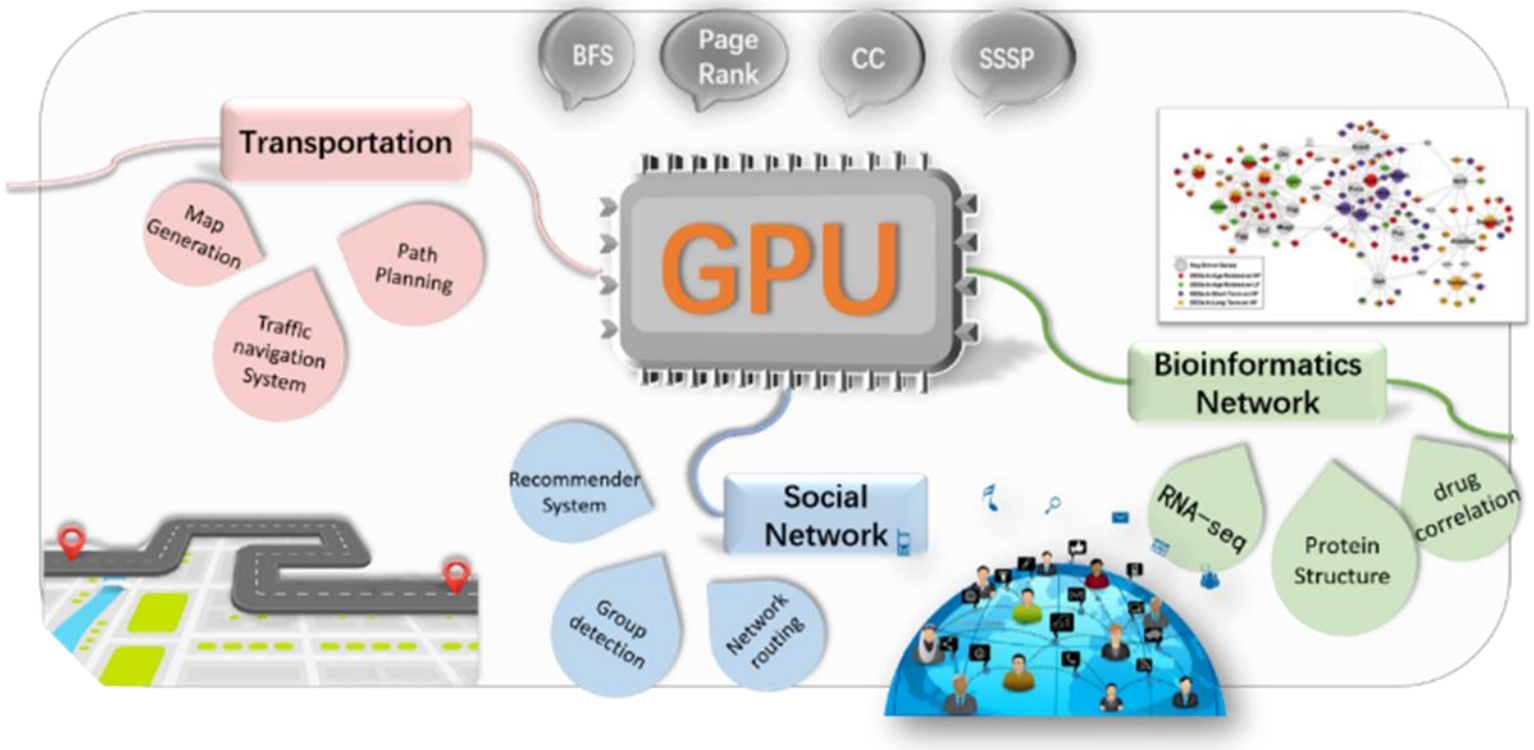
E-mail: eric-li@mail.nankai.edu.cn

南开大学计算机学院 系统与网络研究所

嵌入式系统与信息安全研究室 导师：宫晓利

目录

Out-of-memory Graph Computing



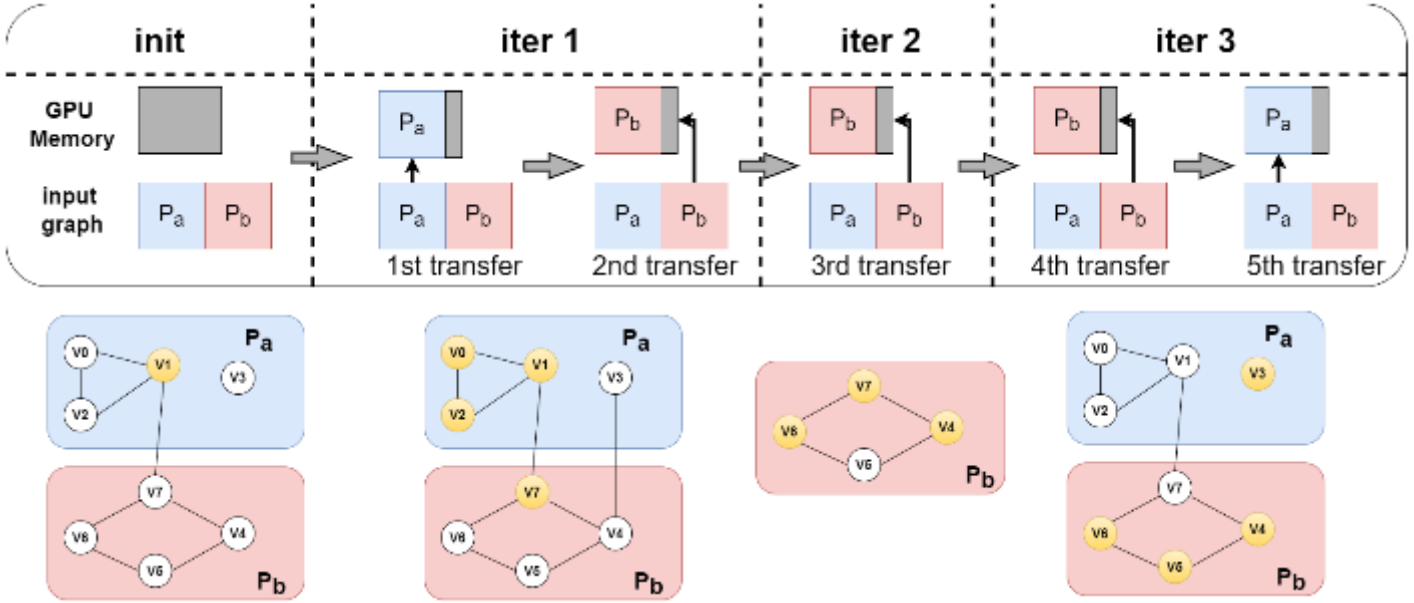
- **图计算**
  - 生物蛋白质分析、社交网络、交通网络 …
  - 数据规模越来越大（上百GB）
  - CPU算不动、GPU装不下

NVIDIA P100	NVIDIA V100	NVIDIA A100	Intel Data Center Max 1550
16GB	32GB	80GB	128GB

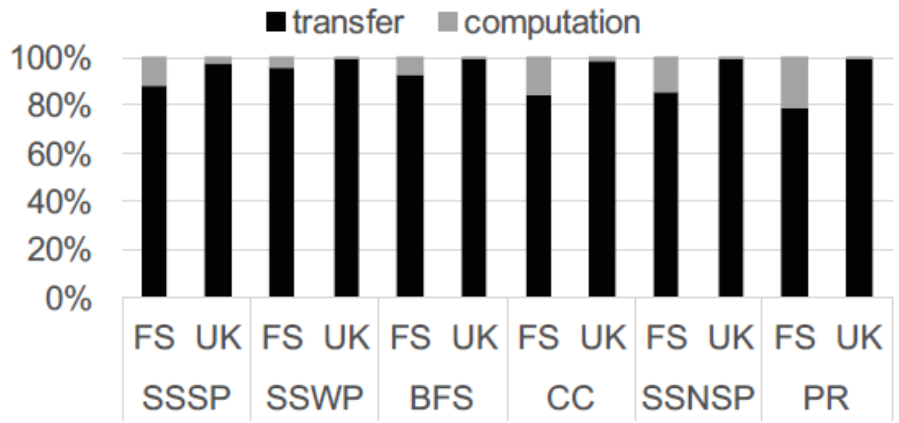
Size	Total	R	P
<100MB	23	12	11
100MB - 1GB	19	9	10
1GB - 10GB	25	9	16
10GB - 100GB	17	5	12
100GB - 1TB	20	8	12
>1 TB	17	5	12

Related works

- **基于CPU-GPU的异构加速方案**
  - 复杂的并发管理与内存管理机制为开发者造成了很大的负担
  - 频繁的主存与显存数据交换易成为性能瓶颈
  - 大量冗余的数据传输
  - 可移植性差、开发难度大（CUDA、ROCm、OpenCL）、跨架构验证难

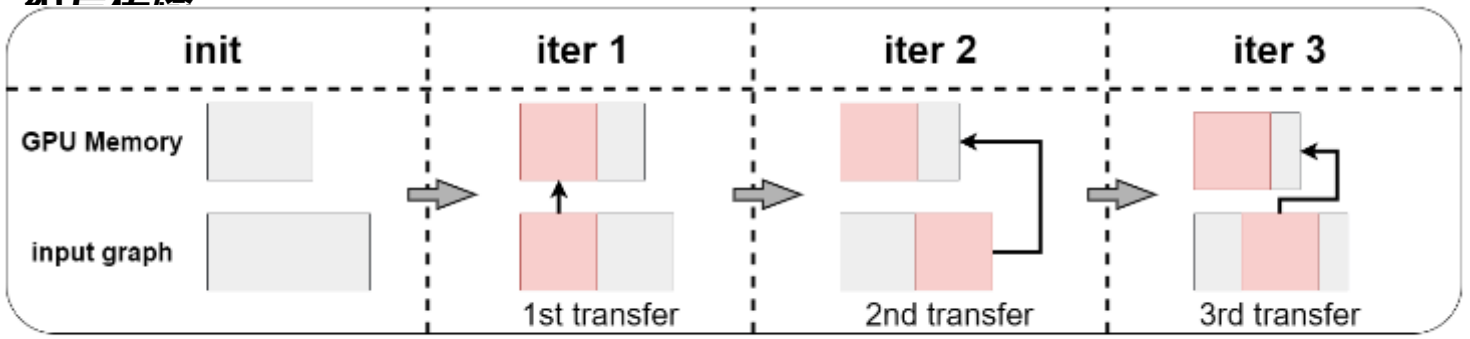


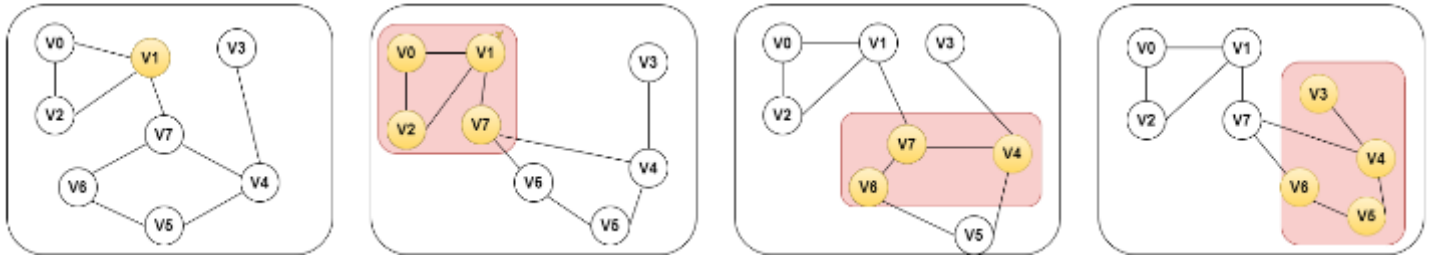
- **Out-of-memory 解决方案**
  - Graph-partitioning
  - Unified Virtual Memory（UVM）



Related works

即使是目前最先进的Graph-partitioning方案（SubWay），仍有50%的GPU时间处于idle，用于等待CPU端的数据组织与传输





Average GPU Memory usage per iteration of SubWay

Dataset	BFS	SSSP	CC	PR
Friendster-konect	0.45GB	0.64GB	1.64GB	2.97GB
UK-2007-04	0.11GB	0.94GB	0.46GB	3.80GB

而基于统一虚拟内存（UVM）的方案，虽然不再需要程序员手动控制数据的传输，但我们的实验结果表明，此时程序需要花费60%的时间等待GPU驱动和OS处理显存中的page fault。在显存占满的时候，频繁的页面换入与换出会导致严重的内存抖动。

事实上，UVM可以被认为是以page为单位的graph-partitioning方法。

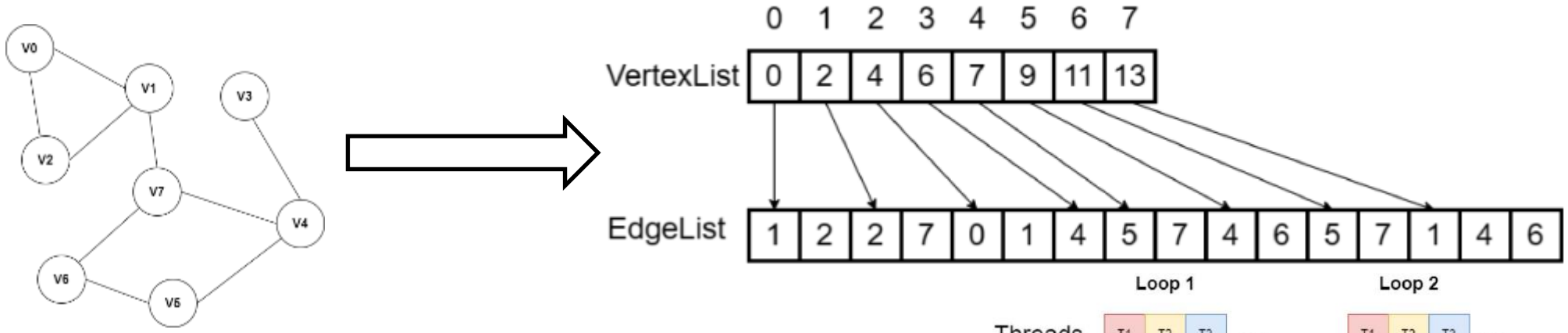
Motivation

三个关键发现:

1. 每一轮迭代事实上只有一小部分数据需要被访问
2. 不同数据粒度下观察到的访存模式不同
3. 每一轮迭代中程序对数据进行大致的线性扫描

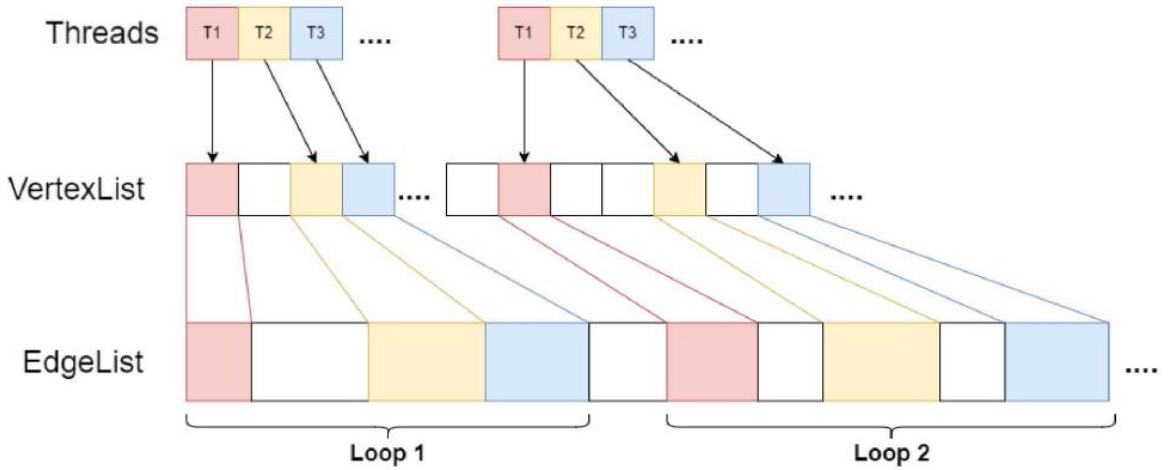
Average active edges per iteration

Dataset	BFS	SSSP	CC	PR
FK	4.5%	3.1%	14.1%	28.7%
UK	0.8%	3.1%	3.0%	25.1%



每一轮迭代过程中，由于可用的线程数总数远小于活跃节点的数量，GPU线程以循环的方式访问图数据。

导致每一轮循环访问的数据一定位于上一轮循环访问的数据之后。

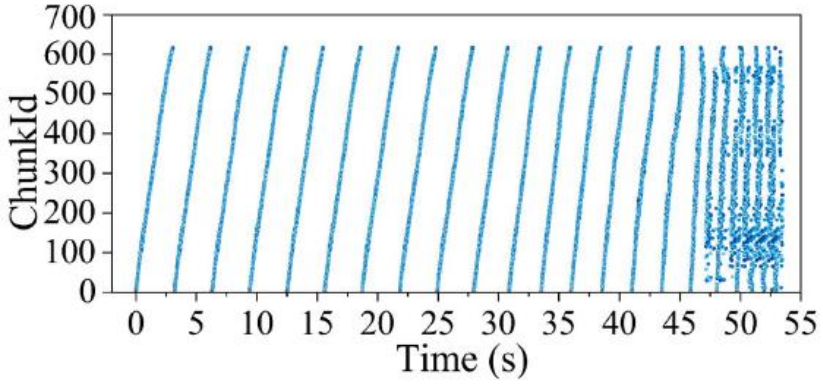


Motivation

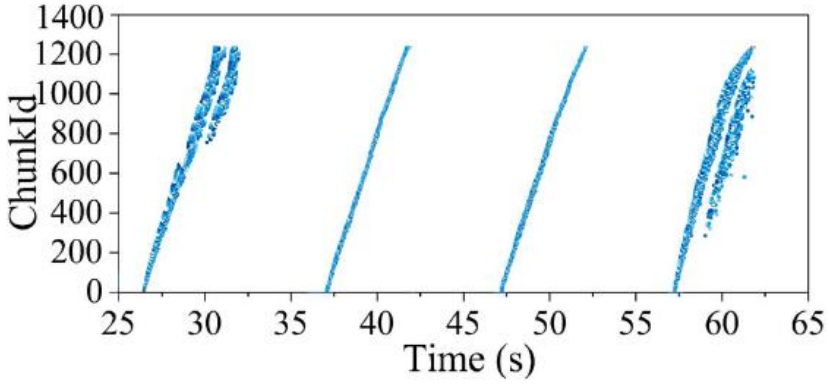
将Edgelist进行分块，每个块包含4百万条边（大约15MB），追踪程序在粗粒度下对数据块的访问规律。

在粗粒度下，这几乎是一个线性扫描的过程，这说明：

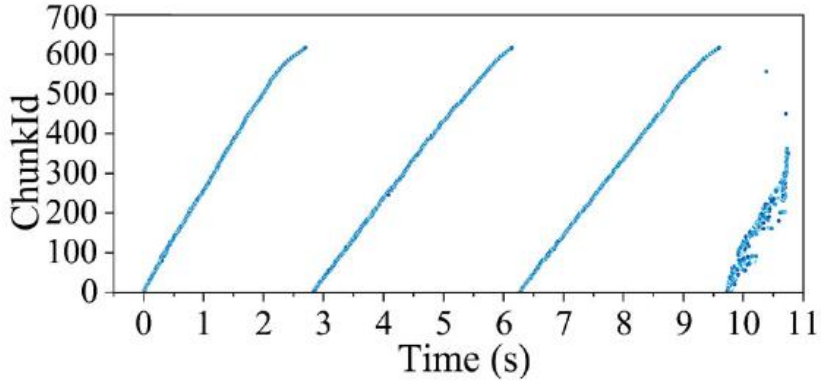
1. 粗粒度视角下，数据在迭代过程中是可以复用的。而基于Graph-partitioning的方法完全无法利用这一点，同时这也说明适当的缓存策略将会是有效的，而且我们完全有足够的显存资源来做缓存。
2. 数据的Reuse distance太长了，导致UVM的LRU换页策略难以捕捉这样的局部性。（LRU的换页策略总是在数据可以被复用之前将数据踢出显存，这也是导致内存抖动的真正原因）
3. 不存在明显的hot spot，所以缓存数据的选取是可以随机的。



(a) PR - Access Patterns at Chunk Granularity



(b) SSSP - Access Patterns at Chunk Granularity



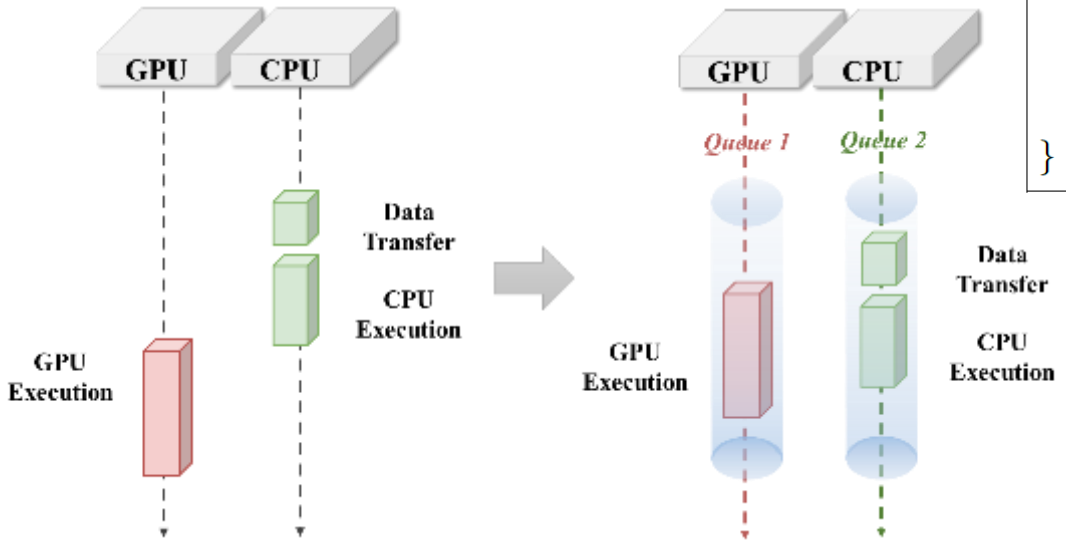
(c) CC - Access Patterns at Chunk Granularity

Opportunities brought by oneAPI

oneAPI的优势

1. 易于上手的异构编程模型
2. 便捷的设备并发管理与内存管理机制
3. 跨架构移植性

基于任务队列的并发管理



VectorAdd in oneAPI

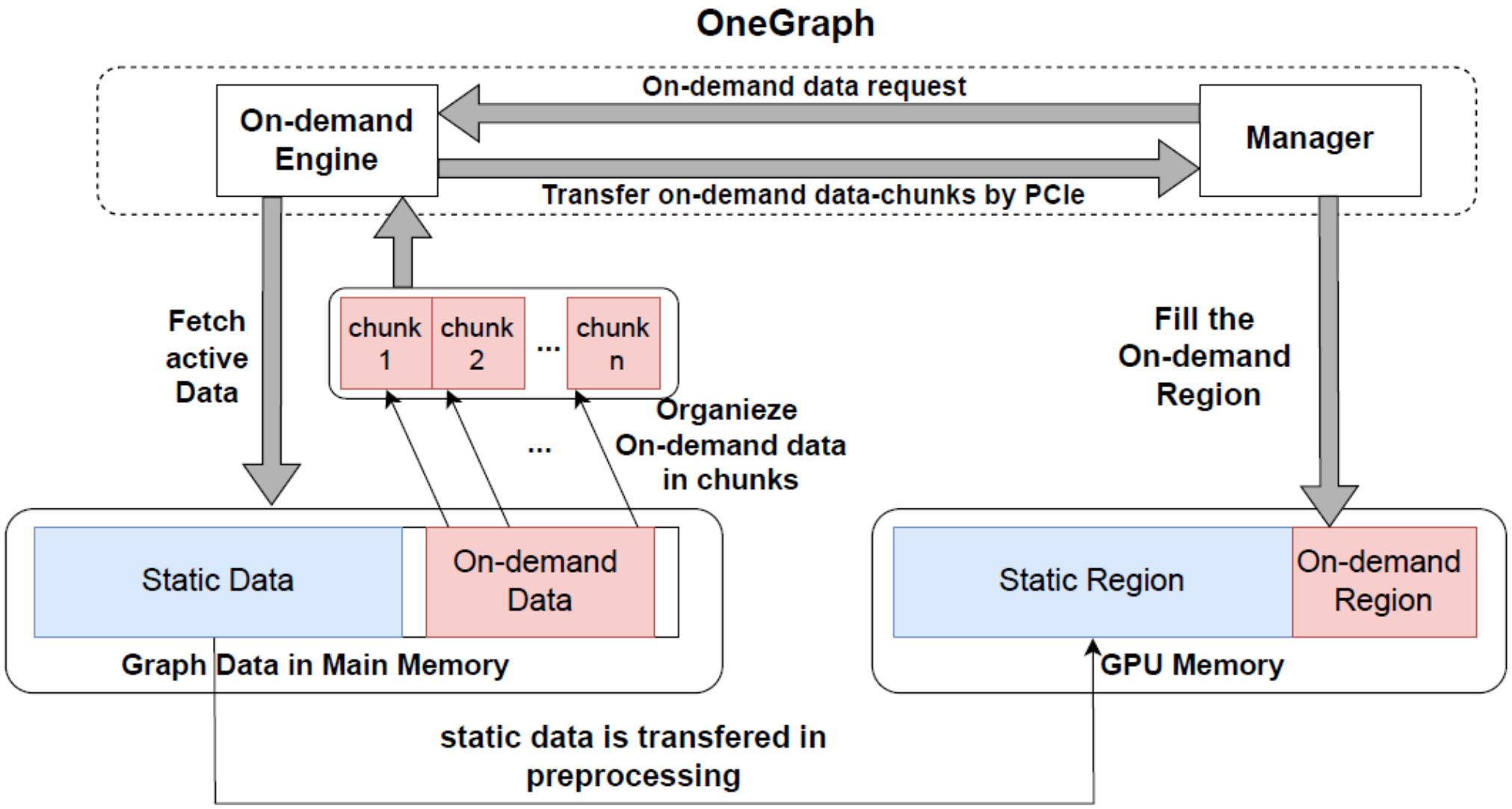
```
//select a heterogeneous platform
auto selector = default_selector;
queue q(selector);//assign a queue to selected
    device
... /*malloc memory for arrays*/
void VectorAdd(queue &q, int *a, int *b, int *
    sum, size_t size){
    q.submmit([&](handler& h) {
        h. parallel_for (size, [=](id<1> tid){
            sum[tid] = a[tid] + b[tid];
        }
    )}).wait();//sync CPU and GPU
}
```

VectorAdd in CUDA

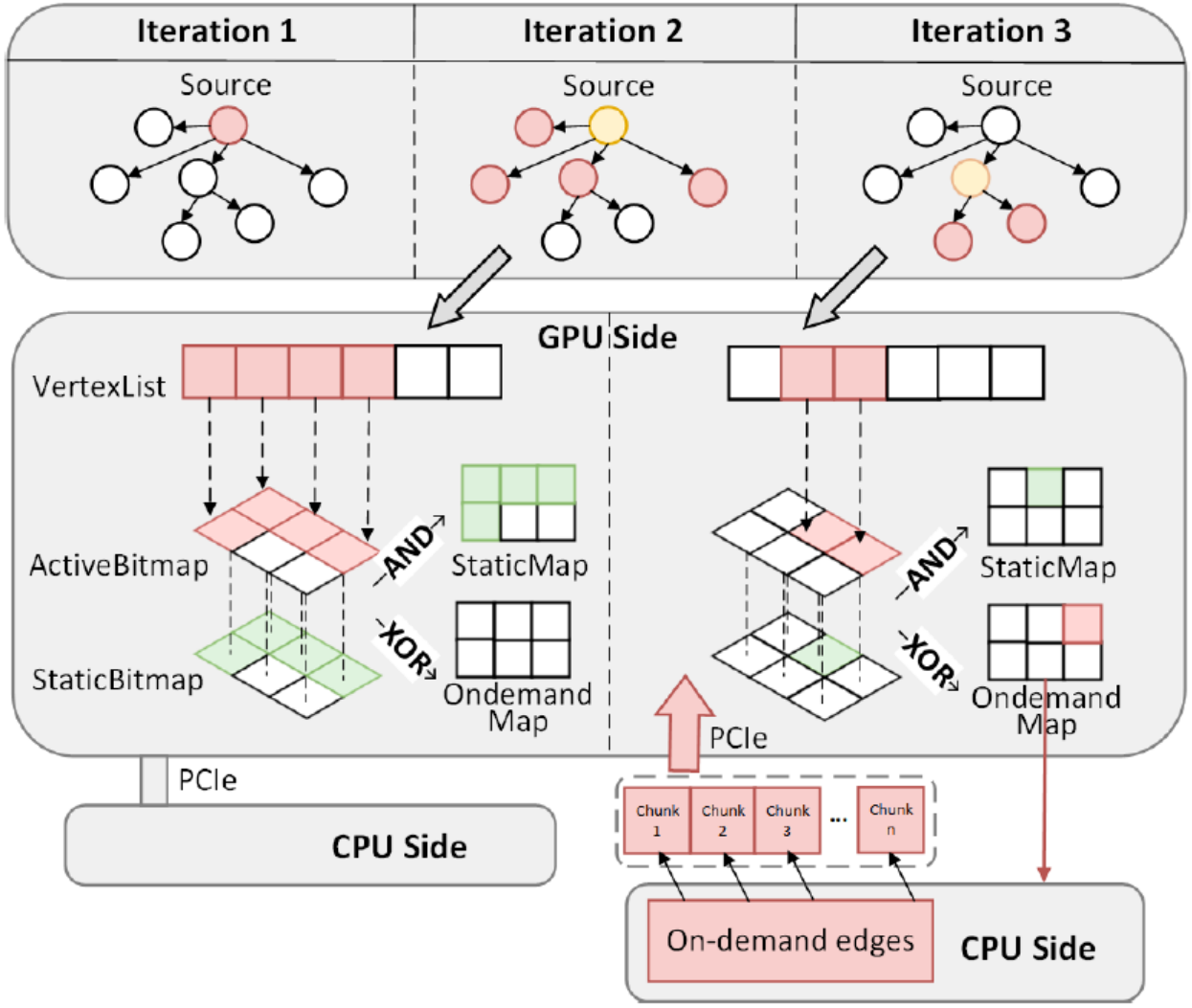
```
//choose an NVIDIA GPU according to
    requirements
cudaDeviceProp devicePropDefined;
memset(&devicePropDefined, 0, sizeof(
    cudaDeviceProp));
cudaChooseDevice(&devicedChoosed, &
    devicePropDefined);
... /*malloc memory for arrays*/
VectorAdd_CUDA(int *a, int *b, int *sum,
    size_t size);
//below is written in another .cu file
"extern C"
__global__ void VectorAdd<<<blocks,threads
    >>>(int *a, int *b, int *sum, size_t size){
    int i = blockDim.x * blockIdx.x +
        threadIdx.x;
    if (i < size){
        sum[i] = a[i] + b[i];
    }
}
void VectorAdd_CUDA(int *a, int *b, int *sum
    , size_t size){
    threads = get_threads_per_block();
    blocks = get_blocks_per_grid();
    VectorAdd<<<blocks,threads>>>(a, b,
        sum, size);
    cudaDeviceSynchronize();//sync CPU and
        GPU
}
```

目录

System Design Overview



Data look up with bitmaps



BFS workflow

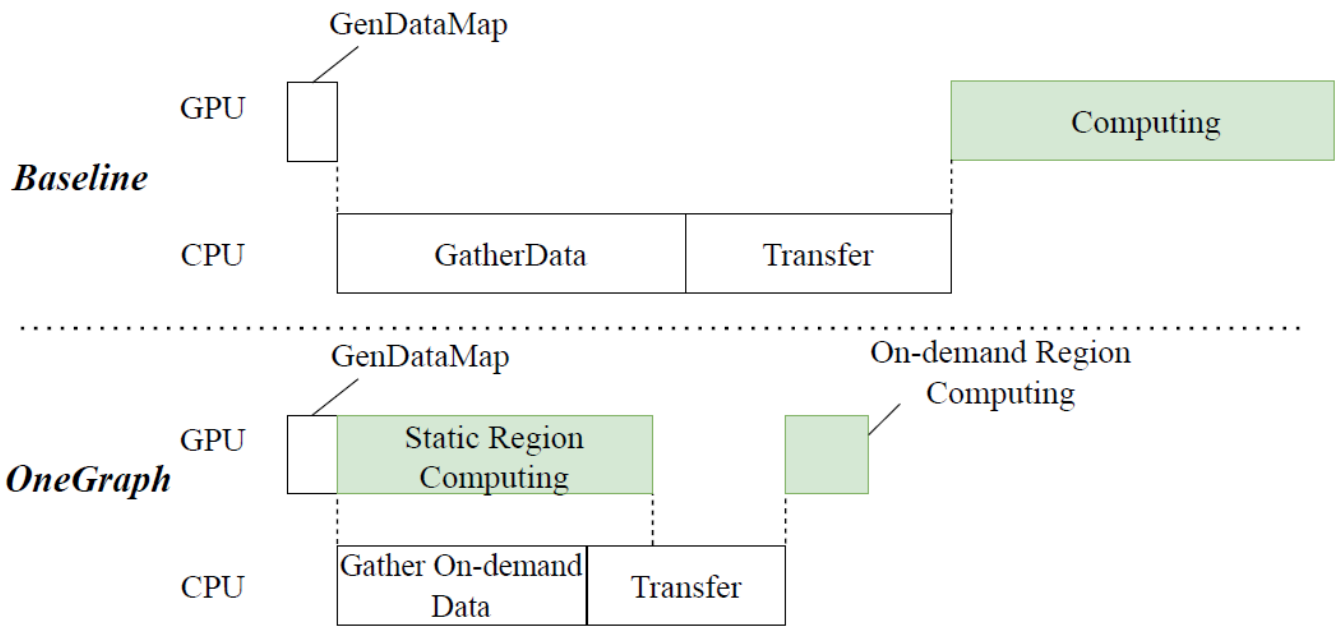
ActiveBitmap：标记活跃节点

StaticBitmap：标记已在显存中的数据

Static Map：标记可以直接开始计算的数据

On-demand Map：标记需要从主存传输的数据

Overlapping of computing and data transfer




```
auto dev_1 = gpu_selector();
auto dev_2 = cpu_selector();
sycl::queue gpu(dev_1), cpu(dev_2);
...
gpu.submit([&](handler& h){
        //StaticRegion computing code on GPU
});
... //on-demand data location code on CPU
cpu.memcpy(dst,src,size).wait();
gpu.wait();//system-wide synchronization
... //On-demand Region computing
```

目录

Experimental setup

	NVIDIA GPU	Intel GPU
Model	A100 PCIe	Data Center GPU Flex 170
Cores	6912	512
Memory	80GB HBM2	16GB GDDR6
Driver	Driver 515.65.01 & CUDA 11.6.2	intel-i915

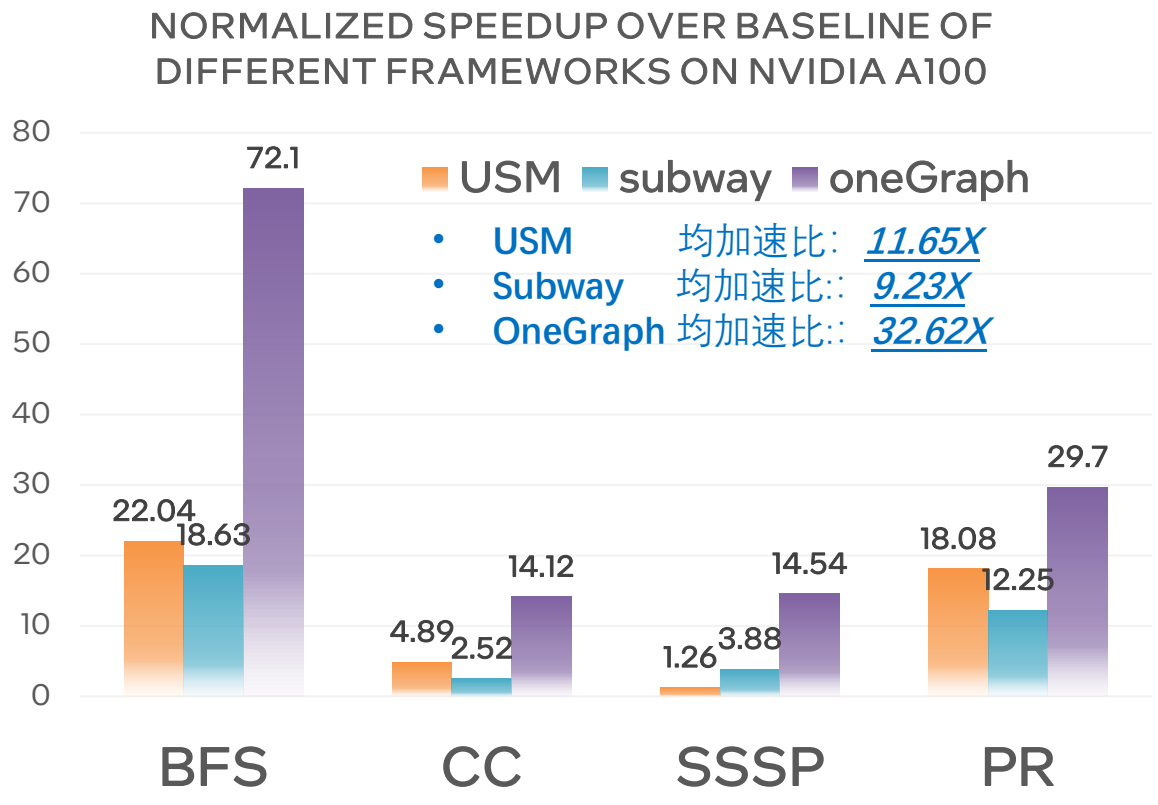
DataSets				
Abbr.	**Name**	**Vertices**	**Edges**	**size**
GS	gsh-2015-host(d)	68.47 M	1.68 B	14G
FK	friendster-konect(u)	68.18 M	2.41 B	11G
UK	uk-2007-04(d)	101.92 M	3.53 B	15G
RMAT1	RMAT-rand(u)	5.25M	1.96B	16G
RMAT2	RMAT-rand(u)	106.67M	3.72B	15G

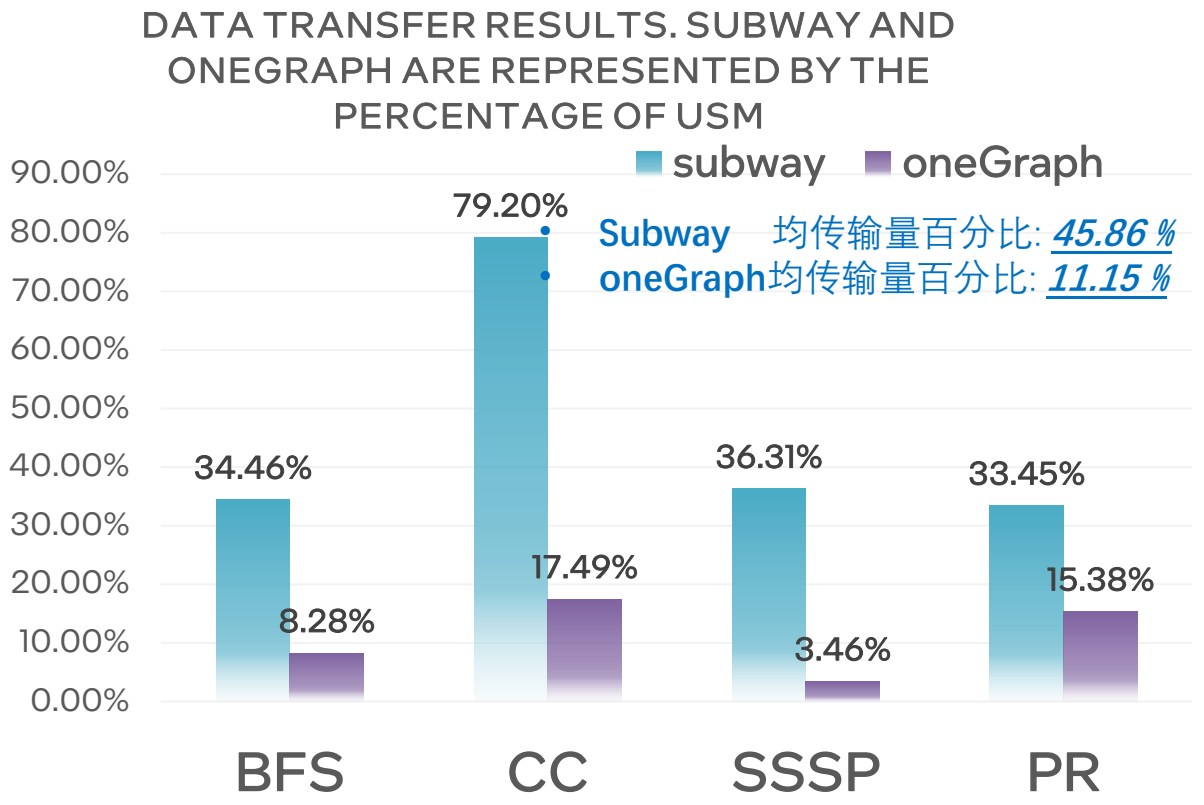
Limiting GPU memory to 12G, graph with edge weight will double size.
Frameworks also will introduce some memory overhead, guaranteeing out-of-mem
Four classical graph algorithms are used, BFS, CC, SSSP, and PageRank.

Experiment results

Comparison schemes (serial algorithm as the baseline)

1. SubWay: the state-of-the-art graph partitioning framework
2. USM: unified shared memory scheme
3. Ascetic: OneGraph in CUDA





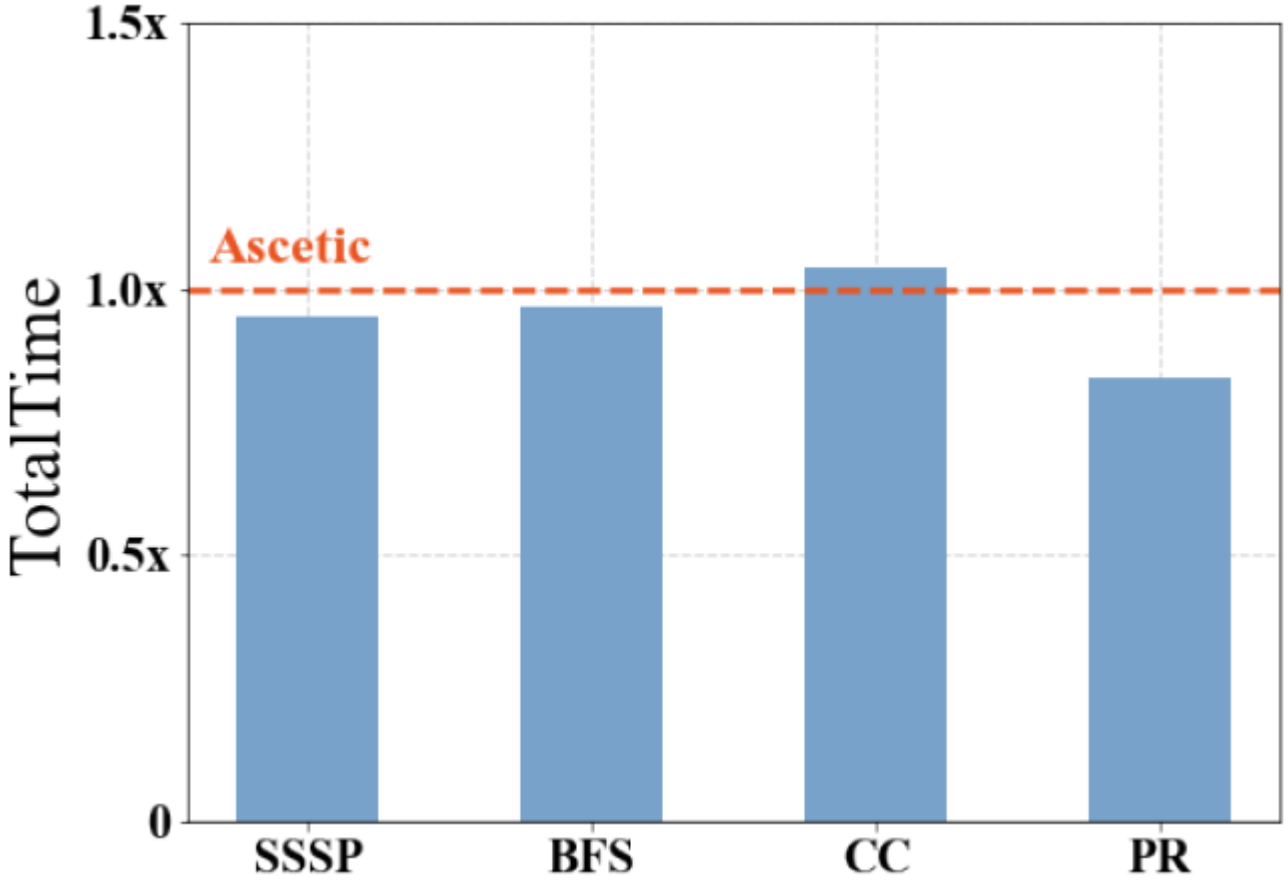
Experiment results

Performance of SubWay and OneGraph on Intel FLEX 170 GPU

	SubWay	OneGraph	SpeedUp
BFS	26.53s	7.98s	3.32x
PR	83.89s	64.43s	1.30x
CC	48.15s	17.93s	2.69x
SSSP	293.4s	56.59s	5.18x
AVERAGE	100.97s	36.73s	2.75x

Comparison with CUDA on NVIDIA A100 GPU

Ascetic is the CUDA version of OneGraph
Performance loss in CC is less than 1%, better performance in other applications



Resources

Papers:

1. [THPC submission] S. Li et al., “OneGraph: A Cross-Architecture Framework for Large-scale Graph Computing on GPUs based on oneAPI”.
2. [IEEE TPDS] S. Li et al., “Liberator: A Data Reuse Framework for Out-of-Memory Graph Computing on GPUs”, in IEEE Transactions on Parallel and Distributed Systems, vol. 34, no. 6, pp. 1954-1967, June 2023.

Code:

The source code of OneGraph is available at https://github.com/NKU-EmbeddedSystem/OneGraph

Home page:

https://scholar.google.com/citations?hl=en&user=KZDQTBQAAAAJ&view_op=list_works&sortby=pubdate

Thank You!
Q&A