Примитивы для обработки графов на GPU

Обзор

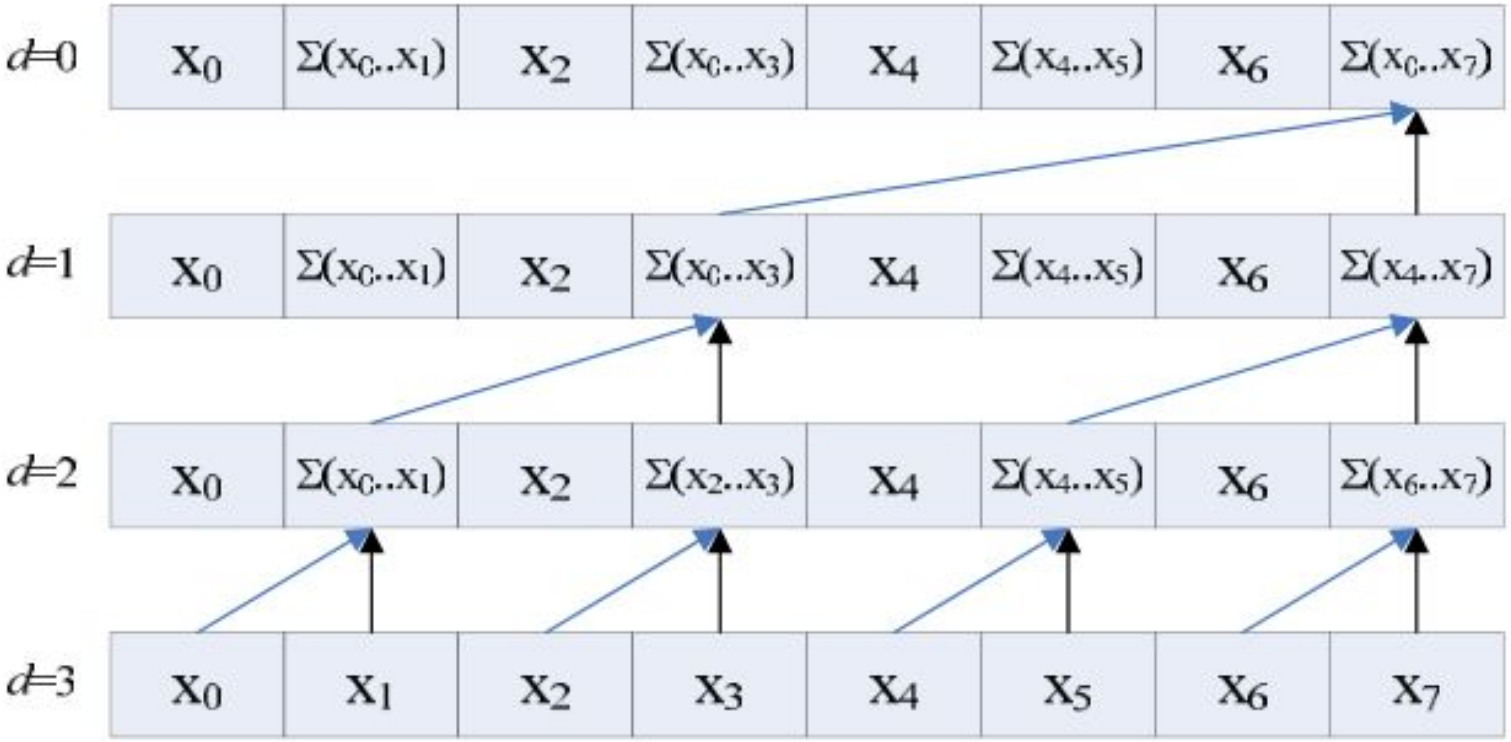
Александр Чебыкин, 371 группа, Матмех СПбГУ
27.04.2017

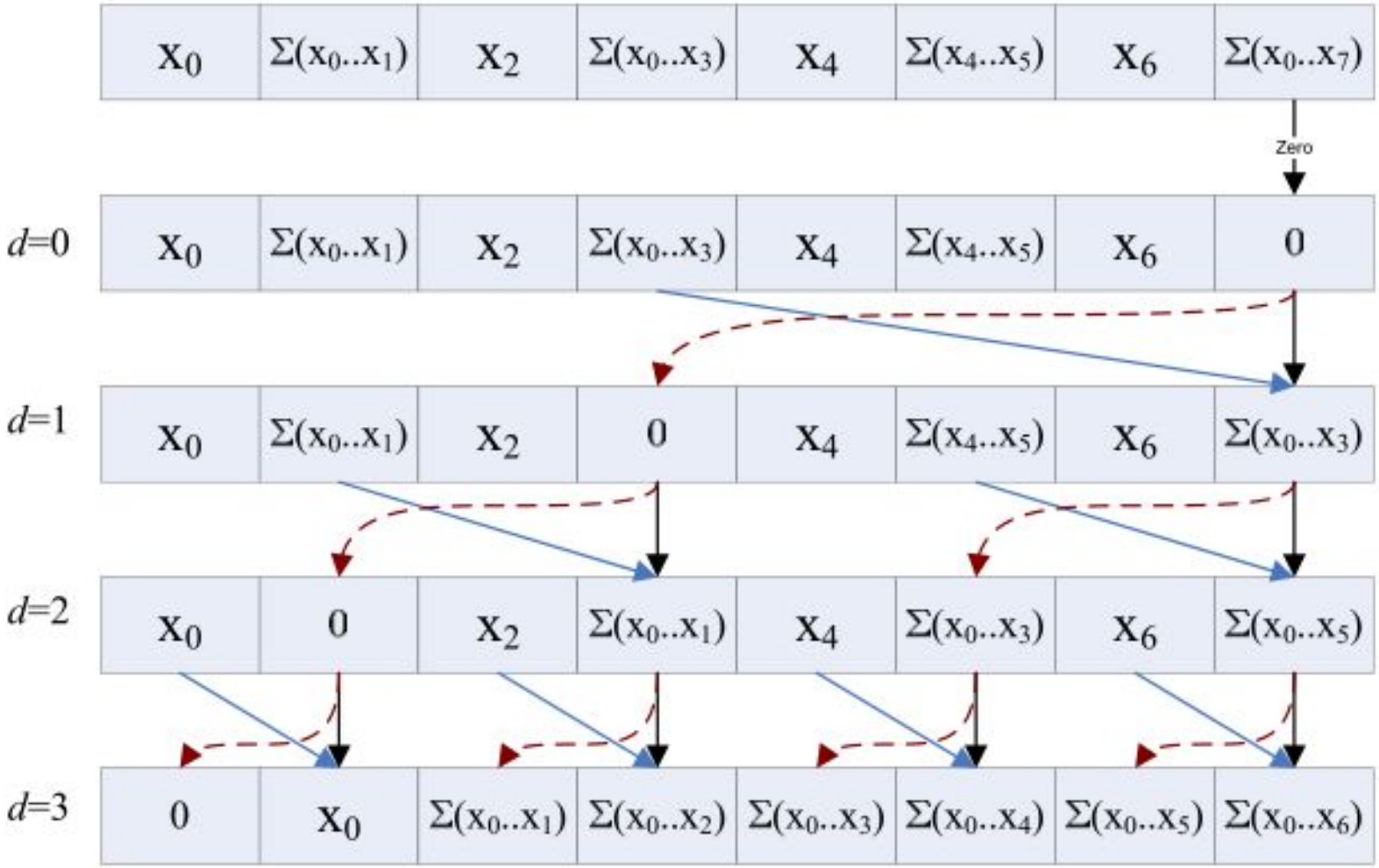
Мотивация

- Программировать на GPU сложно
- Для быстрой обработки больших объемов данных GPU - дешевое решение
- Идея: реализуем переиспользуемую функциональность, скрываем сложность

Первая попытка

- Harris et al, Scan Primitives for GPU Computing, 2007
- scan - алгоритм из параллельных систем, на основе его можно реализовывать другие
- segmented scan
  - делим массив на сегменты
  - внутри сегмента совершаем 2 обхода, похожих на обход бинарного дерева
- Реализация quicksort работает медленнее, чем на cpu, в 2 раза
  - 2007 год
  - авторы заявляют, что не хватает вычислительной мощности
- Используется в будущих библиотеках


d=0
x0
Σ(x0..x1)
x2
Σ(x0..x3)
x4
Σ(x4..x5)
x6
Σ(x0..x7)
d=1
x0
Σ(x0..x1)
x2
Σ(x0..x3)
x4
Σ(x4..x5)
x6
Σ(x4..x7)
d=2
x0
Σ(x0..x1)
x2
Σ(x2..x3)
x4
Σ(x4..x5)
x6
Σ(x6..x7)
d=3
x0
x1
x2
x3
x4
x5
x6
x7


x_0 $\Sigma(x_0..x_1)$ x_2 $\Sigma(x_0..x_3)$ x_4 $\Sigma(x_4..x_5)$ x_6 $\Sigma(x_0..x_7)$
Zero
d=0
x_0 $\Sigma(x_0..x_1)$ x_2 $\Sigma(x_0..x_3)$ x_4 $\Sigma(x_4..x_5)$ x_6 0
d=1
x_0 $\Sigma(x_0..x_1)$ x_2 0 x_4 $\Sigma(x_4..x_5)$ x_6 $\Sigma(x_0..x_3)$
d=2
x_0 0 x_2 $\Sigma(x_0..x_1)$ x_4 $\Sigma(x_0..x_3)$ x_6 $\Sigma(x_0..x_5)$
d=3
0 x_0 $\Sigma(x_0..x_1)$ $\Sigma(x_0..x_2)$ $\Sigma(x_0..x_3)$ $\Sigma(x_0..x_4)$ $\Sigma(x_0..x_5)$ $\Sigma(x_0..x_6)$

Medusa

- Zhong, He, Medusa: Simplified Graph Processing on GPUs, 2013
- Edge-Vertex-Message model
  - Раньше параллелили только по вершинам, теперь по любому из трёх
- Лучше cpu в 1 - 12 раз
- Хуже специализированных алгоритмов на GPU в 1 - 3 раза

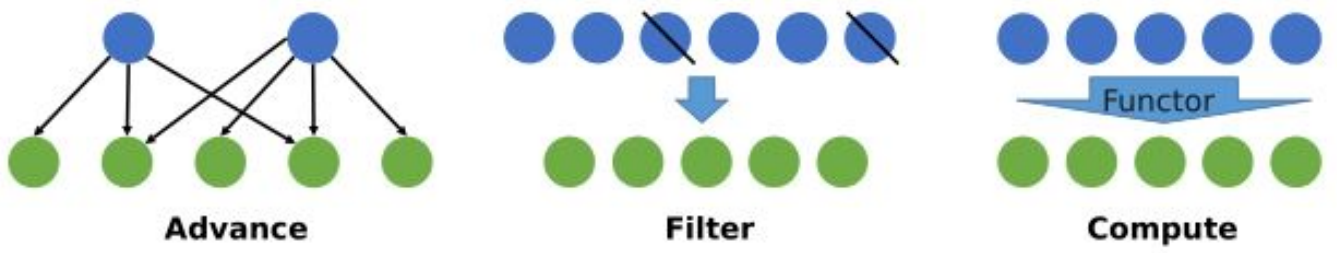
PageRank, Medusa

- elist: отправляем сообщение соседним вершинам
- combiner: комбинация сообщений для каждой вершины
- vertex: обновляем ранг

API Type	*Parameters*	*Variant*	*Description*
ELIST	Vertex v, Edge-list el	Collective	Apply to edge-list el of each vertex v
EDGE	Edge e	Individual	Apply to each edge e
MLIST	Vertex v, Message-list ml	Collective	Apply to message-list ml of each vertex v
MESSAGE	Message m	Individual	Apply to each message m
VERTEX	Vertex v	Individual	Apply to each vertex v
Combiner	Associative operation o	Collective	Apply an associative operation to all edge-lists or message-lists

Gunrock

- Wang et al, Gunrock: A High-Performance Graph Processing Library on the GPU, 2016
- Вычисление в терминах “рубежей” (frontiers)
- Операции над рубежом
  - advance
  - filter
- BFS
  - advance: обновляем метки соседних вершин, используются атомарные операции
  - filter: убираем ненужные вершины



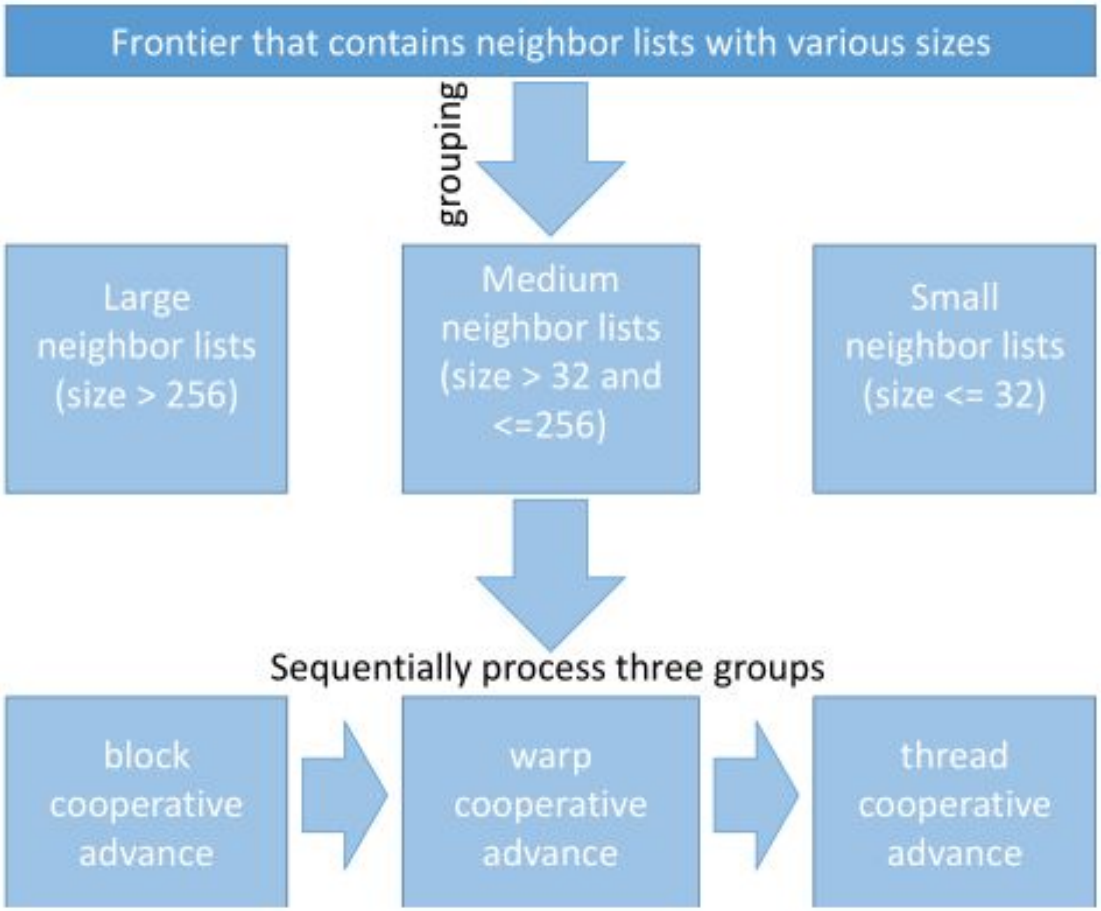
Gunrock, модель программирования

Компоненты:

- problem
  - топология графа
  - специфичный для конкретного алгоритма интерфейс для работы с данными
- functors
  - что делать на одном шаге итерации
  - для filter - CondEdge, CondVertex
  - для advance - ApplyEdge, ApplyVertex
- enactor
  - определяет последовательность шагов из advance и filter.

http://gunrock.github.io/gunrock/doc/annotated_primitives/annotated_primitives.html

Gunrock, load balancing



Gunrock, производительность

- vs Библиотеки для CPU
  - ускорение в 6-337 раз
- vs Специализированные алгоритмы для GPU
  - сравнимая производительность на BFS, Betweenness Centrality, SSSP
  - медленнее на CC
- vs Библиотеки для GPU
  - в среднем быстрее всех
  - в отличие от некоторых библиотек (MapGraph, Medusa), не достигает предела по памяти на тестовом оборудовании на больших графах
  - среднее ускорение по сравнению с Medusa (в разах)
    - BFS: 6.938
    - SSSP: 11.88
    - PageRank: 8.982

Итог

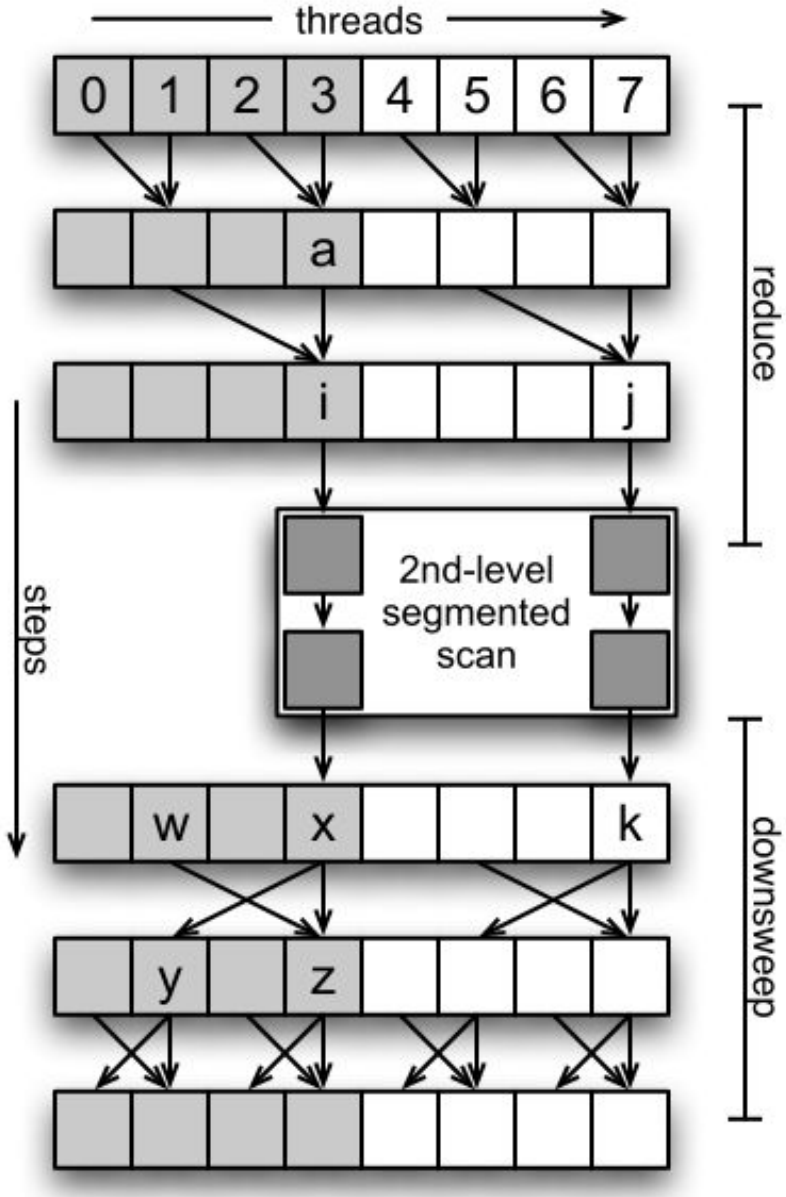
- Gunrock - лучшая библиотека
- Цель - интегрировать Gunrock в QuickGraph

Дополнительные картинки

Segmented scan, QuickSort

```
[5 3 7 4 6]     # initial input
[5 5 5 5 5]     # distribute pivot across segment
[f f t f t]     # input > pivot?
[5 3 4][7 6]    # split-and-segment
[5 5 5][7 7]    # distribute pivot across segment
[t f f][t f]    # input >= pivot?
[3 4 5][6 7]    # split-and-segment, done!
```

Segmented scan



Medusa, PageRank

Device code APIs:

```
/* ELIST API */
struct SendRank{
__device__ void operator() (EdgeList el,
Vertex v){
  int edge_count = v.edge_count;
  float msg = v.rank/edge_count;
  for(int i = 0; i < edge_count; i ++)
    el[i].sendMsg(msg);
}
/* VERTEX API */
struct UpdateVertex{
__device__ void operator() (Vertex v, int
super_step){
  float msg_sum = v.combined_msg();
  vertex.rank = 0.15 + msg_sum*0.85;
}
```

Data structure definitions:

```
struct vertex{
  float pg_value;
  int vertex_id;
}
struct edge{
  int head_vertex_id, tail_vertex_id;
}
struct message{
  float pg_value;
}
```

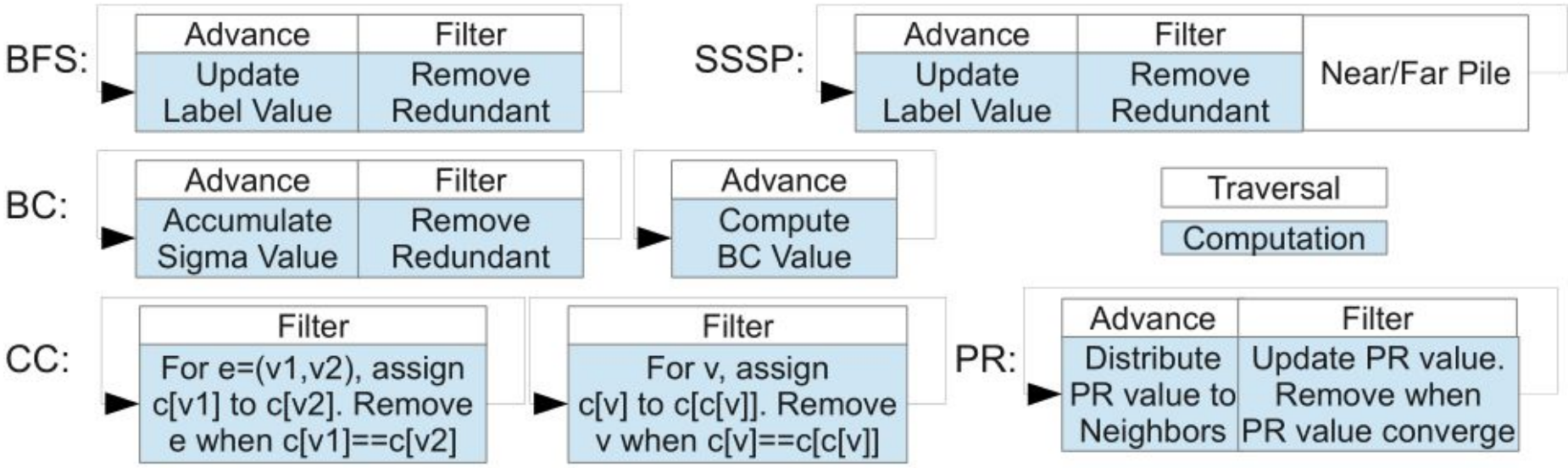
Iteration definition:

```
void PageRank() {
  /* Initiate message buffer to 0 */
  InitMessageBuffer(0);
  /* Invoke the ELIST API */
  EMV<ELIST>::Run(SendRank);
  /* Invoke the message combiner */
  Combiner();
  /* Invoke the VERTEX API */
  EMV<VERTEX>::Run(UpdateRank);
}
```

Configurations and API execution:

```
int main(int argc, char **argv) {
  ......
  Graph my_graph;
  /* Load the input graph. */
  conf.combinerOpType = MEDUSA_SUM;
  conf.combinerDataType = MEDUSA_FLOAT;
  conf.gpuCount = 1;
  conf.maxIteration = 30;
  /*Setup device data structure.*/
  Init_Device_DS(my_graph);
  Medusa::Run(PageRank);
  /* Retrieve results to my_graph. */
  Dump_Result(my_graph);
  ......
  return 0;
}
```

Gunrock, алгоритмы



Gunrock, сравнение производительности