```
func GPMAPlusInsertion (vector<kv> updates):
    updates.sort()
    vector<Segment *> S = bsearch(updates, Global GPMA)
    while ROOT is not in S:
        ~~vector<Segment *~~,
        vector<Pair<Segment *, size_t>> S* = UniqueSegments(S)
        Parallel for seg in S*:
            TryInsertPlus (seg, S*.rightSubVector(), updates)
        if updates.empty():
            return.
        Parallel for seg in S:
            if s.empty(): (Just no update in this seg)
                ~~S.era~~ remove seg From S.
            else:
                (seg in S) = seg.parent
    # still not done:...
    Double the space of ROOT. (maybe, [new root] [ROOT] [new nodes.])
    ~~goto begin~~.
    TryInsertPlus (NewRoot, NULL, U) (Assume, U.size() < OldGPMA.size())
```

answer

~~10000 — 8321 8323~~

~~5k — 3987 3988~~

see

see #5.

① sort (update_keys, values)

② update_nodes.

~~get_leaf~~ locate_leaf_batch fills

(should pass ~~GPMA_overflow~~)

dev.

row Num = 4

DEV offset =

0	0	0	0	0

5

0->NONE, 1->NONE

row_wall =

0<<32	1<<32	2->None	3->None

tmp_value =

1	1	1	1

DEV_VEC keys =

~~None~~ MAX	none	~~None~~ MAX	none

4

DEV values =

1		1	

4

seg_length = 2

tree_height = 1

recale_dense

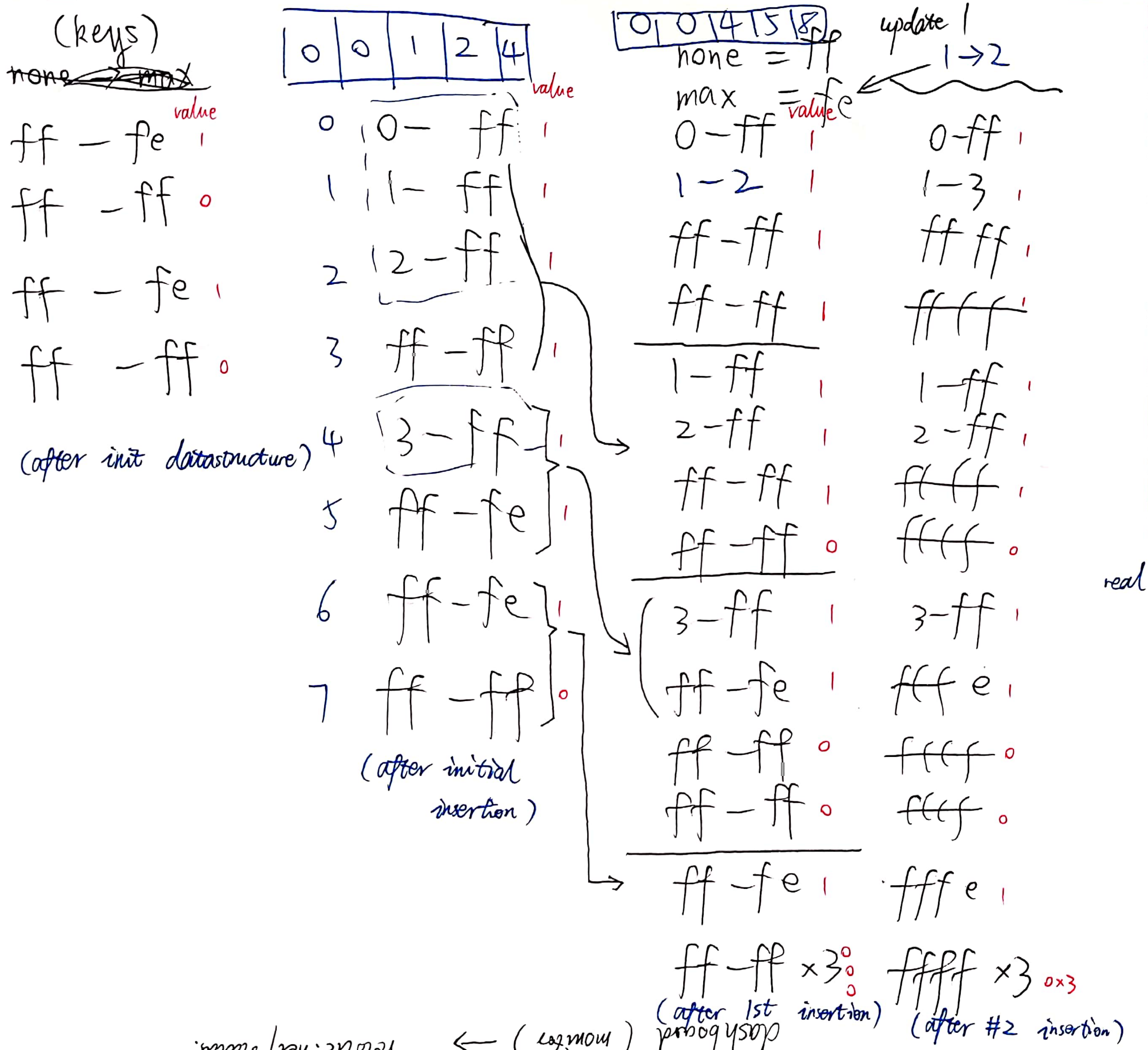
lower_element =

upper element =

(host-only CPU data structure)

Leaf. +1 Level 1

1	2	3	4
11	21	31	
12	22		
13			
14			
15			
16			
17			



real update 2: 1→3, 1→2.

① (add) ⓪ (delete)

dataset

4	2
1 →	2
1 →	3

dashboard (monitor) → recoliz.net/status.

services:

#3

ff - fe

ff - ff

ff - fe

ff - ff

(same)

update: [0 → 1 , val 1 ; 0 → 3 , 1]

	key	value
0	0 - 1	1
	0 - 3	1
	ff - ff	1
	ff - ff	1
4	0 - ff	1
5	1 - ff	1
	ffff	1
	ffff	1
8	2 - ff	1
9	3 - ff	1
	ffff	0
	ffff	0
	fffe	1
	fffe	1
	ffff	0
	ffff	0

offset

0	4	5	8	9

update: [1 → 2 , 1 ; 2 → 3 , 1 ; 0 → 1 , 0 ; 0 → 3 , 0]

	key	value
0	0 - ff	1
	1 - 2	1
	ff - ff	1
	ff - ff	1
4	1 - ff	1
	2 - 3	1
	ff - ff	1
	ff - ff	1
8	2 - ff	1
9	3 - ff	1
	ffff	0
	ffff	0
	fffe	1
	fffe	1
	ffff	0
	ffff	0

offset

0	0	4	8	9

edge. From 0 ~ 1000 : GPU

1000 ~ 2000 : CPU

```
GPMA<CPU> g1;
GPMA<GPU> g2;//(single GPU)
- - - - - - - - -
GPMA<GPU> g3(GPUID::_2);
                // multiple GPU.
```

constant:

None = ff

MAX = fe

#4

index	update_width.	<ThreadPerBlock, itemPerthread>		<<<block N, thread N>>>
0	2	2	1	1 (unique_update size.) 2
1	4	4	1	4
	8	8	1	8
	16	16	1	16
	32	32	1	32
	64	32	2	32
	128	32	4	32
	256	32	8	(var) 32
	512	32	16	32
	1024	32	32	32

one update per block.

unique.

(new dataset).

tmp: answer

I generated this dataset at commit 876686a3. No the dataset was shuffled ~~Maybe script changed~~ on every run.

10k.	8318 → 8261	SHA256 Hash = 1ea2
5k	3984 → 3924	hash = 4dec
10	5 → 5	hash = 6e6c
FULL (CPU)	1333870 → 1334630.	hash = eb1e

commit 876686a3

32 · CPU

RAM 1.14G

verified: correct.

verified at P4 (API61)

GEF 750 (API50)

commit 876686a3

CPU (P4) ~~11.17~~ (first trial) user, 1.22 system.

1Core CPU 740MB RAM.

1333870

(several trials) 6.95 user 1.14 system

1334630

CPU (4Core) 486 user, 22.35 system, 368% cpu.

100% × 4 CPU, 1.78GB RAM. (modified gpma, disable blocked).

#5

DEV

results. size = nodeSize.

bitmap. size = ceil($\frac{\text{nodeSize}}{32}$) { nodeSize , bits.
init = {0}.

nodeQ. size = nodeSize
* offset = 1

edgeQ size = ~~node~~ edgeSize. [edgeQ SHOULD be named as neighborNodeQ.]
* offset = 0

CPU ~~host_num = start_node.~~

nodeQ [0] = ~~host_num~~ start_node

bitmap[start_node /32] = 1 << (start_node %32)

setbit (start_node.)

nodeQ. offset = 1

results [start_node] = 1

CPU: level = 1 , tmp = 1

while True:

edgeQ. offset = 0 — clear neighborQ.

gatherK. <<< ~~f(tmp)~~ nodeQ. offset. >>> — fill neighborQ basing on nodeQ

~~work on nodeQ~~ ++level. — ++level.

nodeQ. offset = 0 — clear nodeQ.

~~tmp = edgeQ. offset~~

~~work on edgeQ~~ contractK. <<< ~~f(tmp)~~ edgeQ. offset >>> — fill nodeQ basing on neighborQ and fill results[].

~~tmp = nodeQ. offset~~

if nodeQ. offset == 0 , break. — exit if no more NEW nodes.

}

contract: update bitmap.

contractK: //Summarize: For every node in edgeQ, set results[node] = level for ONLY NEW nodes, and put these NEW nodes into nodeQ.

<<< edgeQ.offset >>>

~~parallel for i from 0 to edgeQ.offset~~

parallel for neighbor in edgeQ:

strange cache... not fully understood.

Todo, understand the cache speed-up.

```
strange warp_cache code; (not-able to understand) //TODO.

if neighbor in cta_cache:
    ignore it. (end the BIG for)
else:
    add neighbor to cta_cache

if neighbor in bitmap is set:
    ignore it. (END BIG FOR)
else:
    bitmap[neighbor].set().

if results[neighbor] is not empty: //This node has already touched
    ignore it( )
else:
    set results[neighbor] = level.

push neighbor to nodeQ.
```

#

gatherK: // Summarize: For every node in nodeQ, push its neighbors to edgeQ.
<<< nodeQ.offset >>>

(ignore NULL)
(include visited)

∞ #

~~for i~~

parallel for i from 0 to nodeQ.offset:

node = nodeQ[i]:

row_begin/end = [row_offsets[node], ... [node+1]]

SAME block code:

PR

while someone has data with len >= ThreadsInBlock:

shared row_begin_row_end = random one;

Now, threads in this block solves the winner's data:

(block) parallel for gather from shared_row_begin to shared_row_end:

key[gather], value[gather]

Edge.TO = Low32b(key[gather]) // neighbor node here!

continue if Edge.TO or value[gather] is NULL.

edgeQ.offset += howManyTask In This Block This Turn.

edgeQ[edgeQ.offset + MyOffset In This Turn] = Edge.TO.

// EdgeQ.push(Edge.TO) neighbor node

// Now everyone has len < ThreadsPerBlock.

DO PR Again on warp-wide. (meaning that ThreadsIn~~Block~~Warp = 32).

// Now everyone has len < 32. (WarpSize)

DO something similar to PR again on element-wide. Refer to #9 for detail.

return.

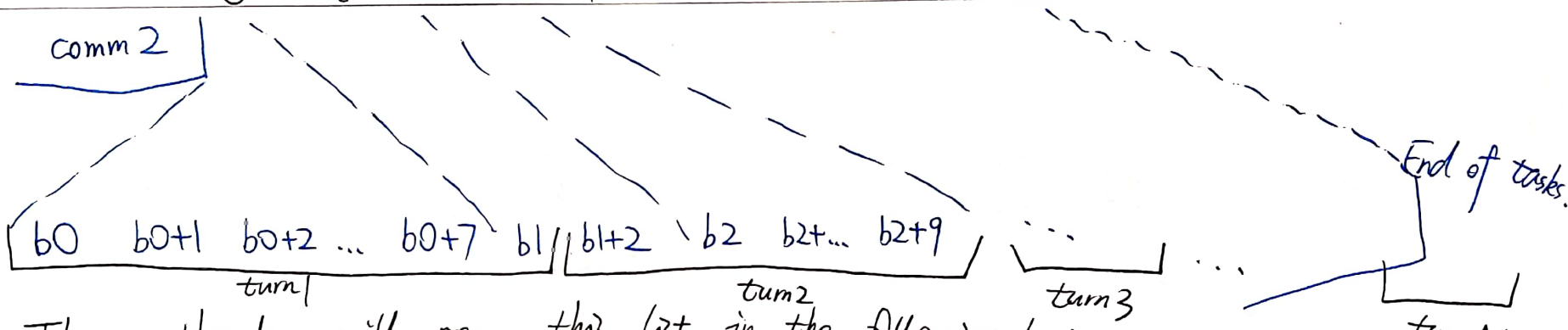
#8

#8

threads:

threads.

thread_data	8	2	9	3	2	3	1	5	
rsv_rank	0	8	10	19	22	24	27	28	total = 33.



Then threads will process this list, in the following logic:

```
gather = comm2[i]
Edge.TO = Low32b(keys[gather])
...
// Refer to #8 Red code...
```

END

Performance:

~~Todo: Allow multi-CPU~~ ☆.

Done.

The Machine in this page:

intel Xeon Cascade Lake
2.5GHz, 8 Core, 32GB RAM
1 NviDIA T4, 7Gbps LAN.
(Tencent GN7.2XLarge32)

1GPU 8.87u 1.18s 8.9sE.
Wall time: 23:49 – 23:57 ~~178%cpu~~

1 CPU 58.4u, 23.4s, 157%cpu
Wall time: 24:52 – 25:43

cpus=5

7 GPU

1 CPU

0 1 2 3 4 5 6 7 8 9 10

5
23:49

4
6

58.4
8

Origin: 5.9u, 0.8s, 6.74e. 99%cpu.

All results in this page 并行度不足
did't exclude BFS. Do Not use them..

8CPU: 63u, 2.0s, 24.74e.

7CPU+1GPU 40.13u, 2.46s, 18.46E
f=7
Wall time: 01:48 – 02:07

ERROR: You must
① move dataset to "/tmp"
② DISABLE_BFS.

f=15: 32.9u, 16.8e, 2.58s.
f=40:

#10

Local bad CPU machine: (Machine 1 in report).

(CPU i5 6500 4×3.6GHz)

GTX 750 is too bad...

Unable to run FULL test... use 100000, limit 700 MB. GDDR5

GTX750, factor ~ time(E) 2.82s, CPU×3, GPU×1

F=0, time = 1.85s

F=1, time=1.65s 1.42s

2	1.62s	1.45s
3	1.66s	1.44s
4	1.60s	1.37s
5	1.90	1.77s
6	1.88	1.65s
7	1.76	1.69s
8	1.84	1.61s
9	1.80	1.59s
10	1.74	1.63s
11	1.74	1.69
12	1.79	1.72
13	1.77	1.67
	1.80	1.66
	1.81	1.72
16	1.88	1.84
17	1.89	1.78
18	1.93	1.87
19	1.92	1.83
20	2.03	1.86

NO BFS Result in BLUE

dataset 9999- from /tmp.

		Removed BFS
1 CPU:	47.63s, 110% CPU	43.08, 111%
2 CPU:	32.51s, 173% CPU	25.79, 192
3 CPU:	29.12s, 222% CPU	22.70, 261
4 CPU:	25.46s, 250% CPU	~~18.68s, 307% CPU~~ 18.55s, 308%
5 CPU:	33.38s, 209% CPU	24.32, 251%
6 CPU:	× not tested.	22.66, 270%
7 CPU:	NT	21.47, 287%
8 CPU:	NT	20.89, 305%

#11

(CPU Xeon 8255C 8×2.5GHz) Re-test the T4 GPU platform. (as #11)

Tesla T4, CPU×7, GPU×1. 10GB GPDR6.

NO BFS, F=0, time = 28.97s → note: why 222% CPU? NO BFS

F=1, time = 23.23s 206% CPU

2	21.03	199% CPU
3	19.77	193% CPU
4	18.82	188% CPU
5	18.03	184% CPU
6	18.47	184%
7	17.61	178%
8	17.01	174%
9	16.54	172%
10	16.06	170%
11	15.79	168
12	15.64	167
13	15.50	169
14	16.23	161
15	15.72	158
16	15.64	157
17	15.81	157
18	15.31	155
19	15.50	155
20.	15.32	154

dataset 9999_

from /tmp

NO_BFS

~~1CPU 57.87s, 103% CPU.~~

~~2CPU 38.16s, 159%~~

~~3CPU 35.42, 202%~~

forgot to Remove BFS.

4CPU	1CPU	50.53s, 104% CPU.
5CPU	2CPU	32.16s, 170%
6CPU	3CPU	28.70s, 223%
7CPU	4CPU	26.51s, 231%
8CPU	5CPU	21.62s, 273%
	6CPU	20.58s, 291%
	7CPU	20.05s, 307%
	8CPU.	19.60s, 314%

#12