Evaluating Coprocessor Effectiveness for the Data Assimilation Research Testbed

Ye Feng

IMAGe DAReS SIParCS

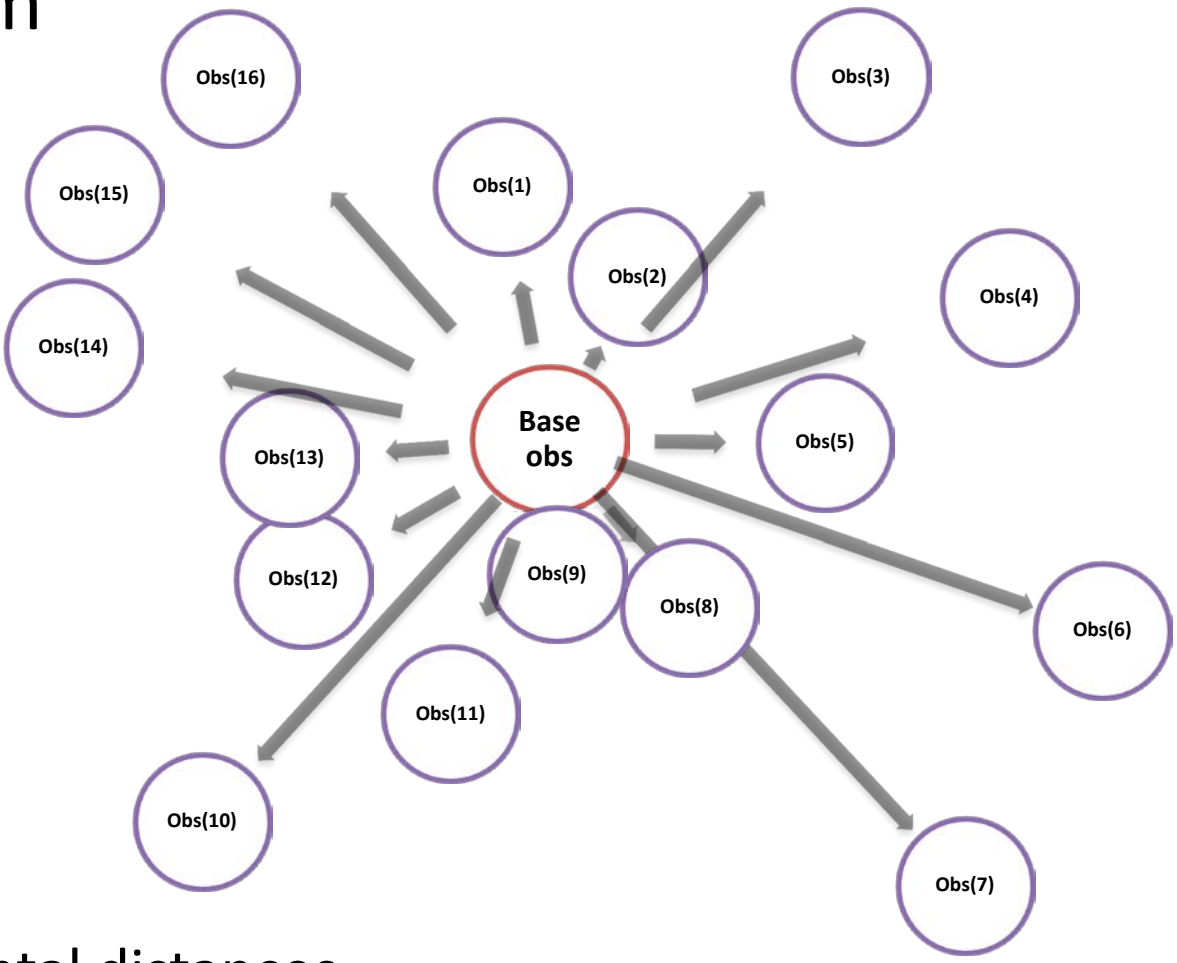
University of Wyoming

Introduction

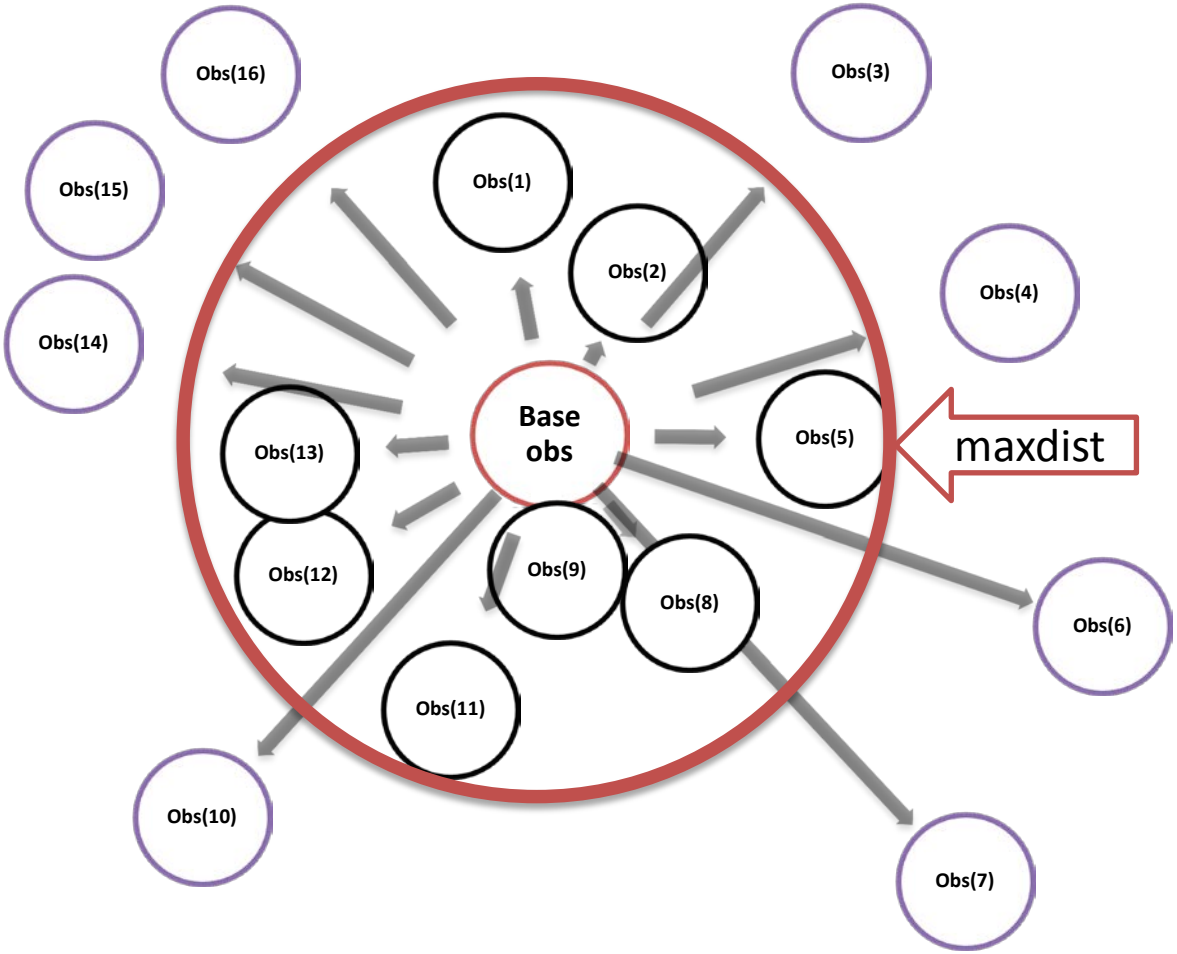
- **Task**: evaluating the feasibility and effectiveness of coprocessor on DART.
- **Target**: get_close_obs

  (profiling result: computationally intensive & executed multiple times during a typical DART run.)
- **Coprocessor**: NVDIA GPUs with CUDA Fortran.
- **Result**: Parallel version of exhaustive search on GPU is faster.

Problem

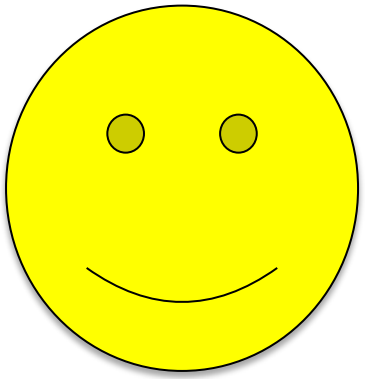


Calculate:
the horizontal distances
between base location and observation locations.



Find:
the close observations.

cclose_ind	cdist
1	d1
2	d2
5	d5
8	d8
9	d9
11	d11
12	d12
13	d13



EASY!
or is it?

It is easy on CPU

```
!-------------------------------------------------------------
! For validation, it is useful to be able to compare against exact
! exhaustive search
if(COMPARE_TO_CORRECT) then
   cnum_close = 0
   do i = 1, gc%num
   this_dist = get_dist(base_obs_loc, obs(i), base_obs_type, obs_kind(i))
      if(this_dist <= this_maxdist) then
         ! Add this obs to correct list
         cnum_close = cnum_close + 1
         cclose_ind(cnum_close) = i
         cdist(cnum_close) = this_dist
      endif
   end do
endif

!-------------------------------------------------------------
```

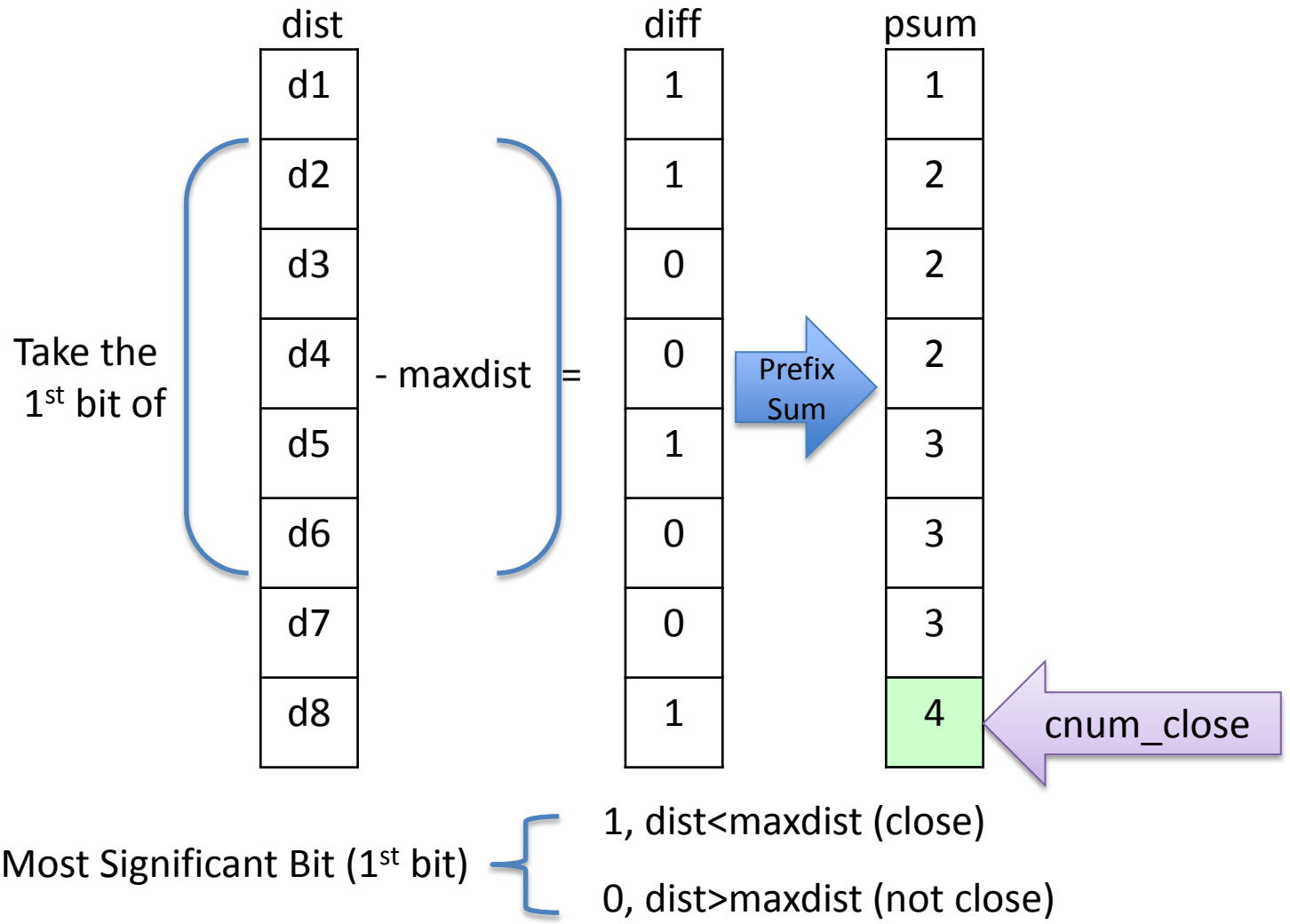
Data Dependency

But GPU doesn't work this way!

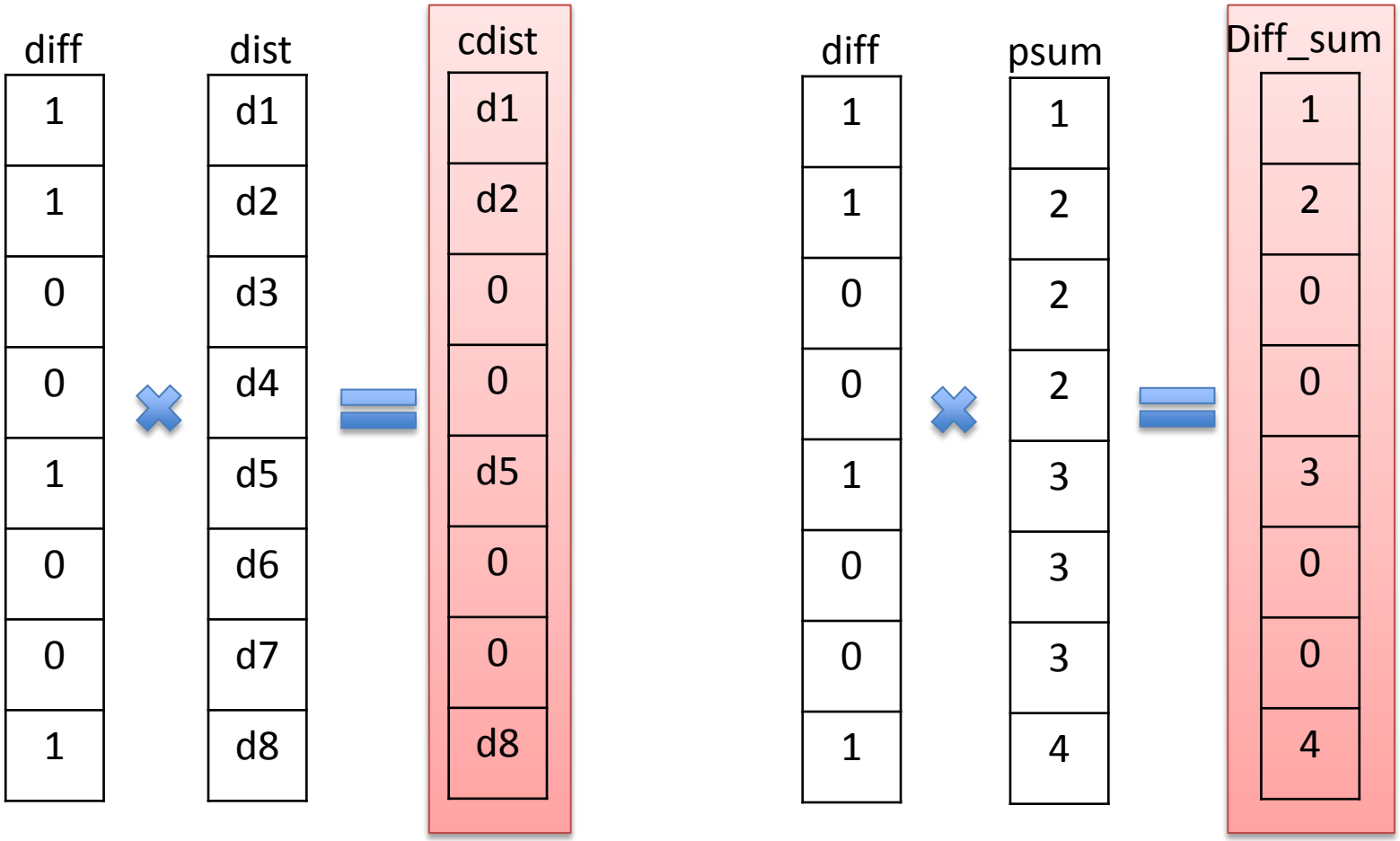
Problems with data dependency usually don't scale so well on GPU.

- *cnum_close depends on previous cnum_close value.*
- *cclose_ind and cdist both depend on cnum_close.*

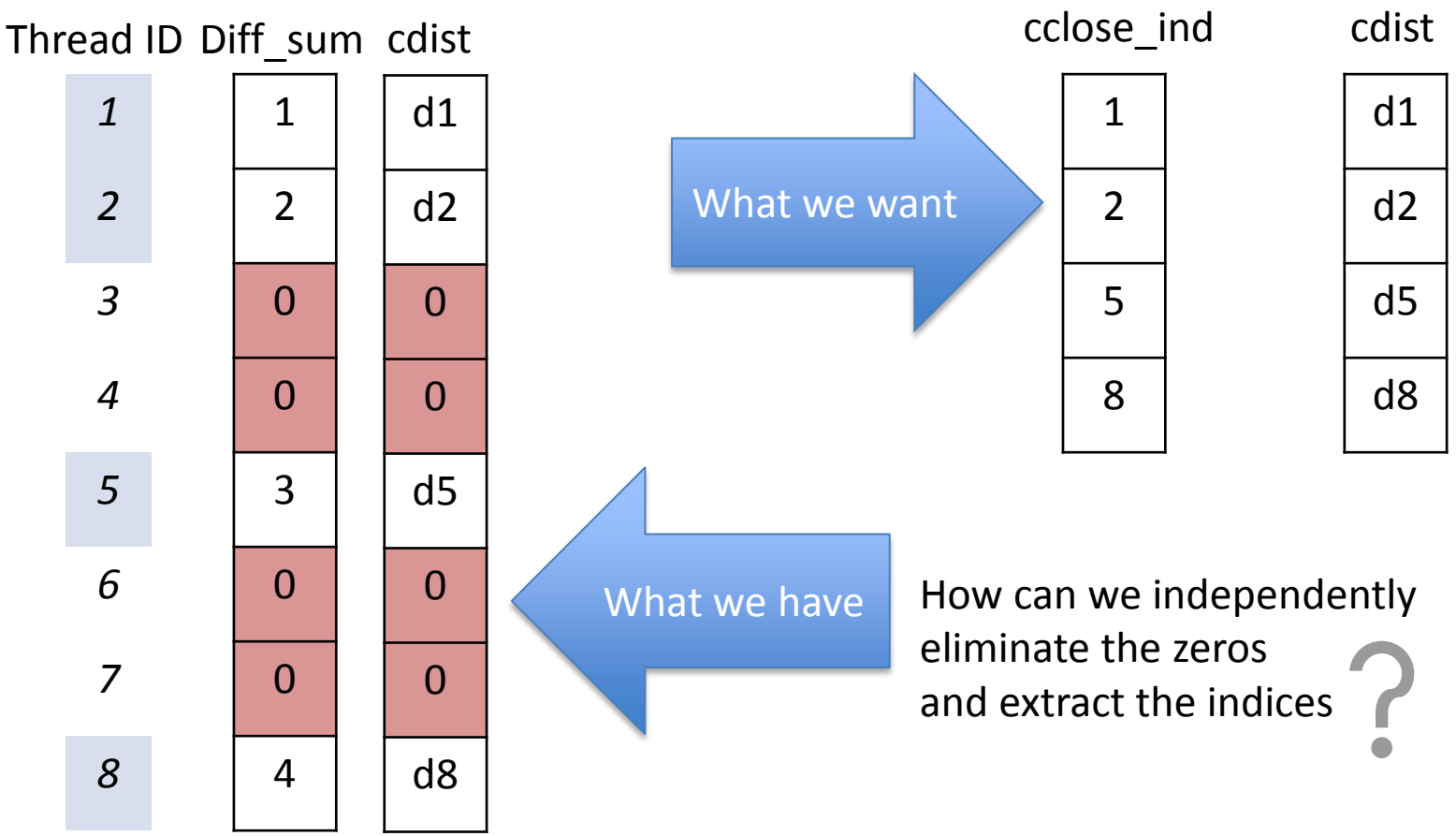
GPU Scan:


dist
diff
psum
d1
d2
d3
d4
d5
d6
d7
d8
Take the 1st bit of
- maxdist
=
1
1
0
0
1
0
0
1
Prefix Sum
1
2
2
2
3
3
3
4
cnum_close
Most Significant Bit (1st bit)
1, dist<maxdist (close)
0, dist>maxdist (not close)

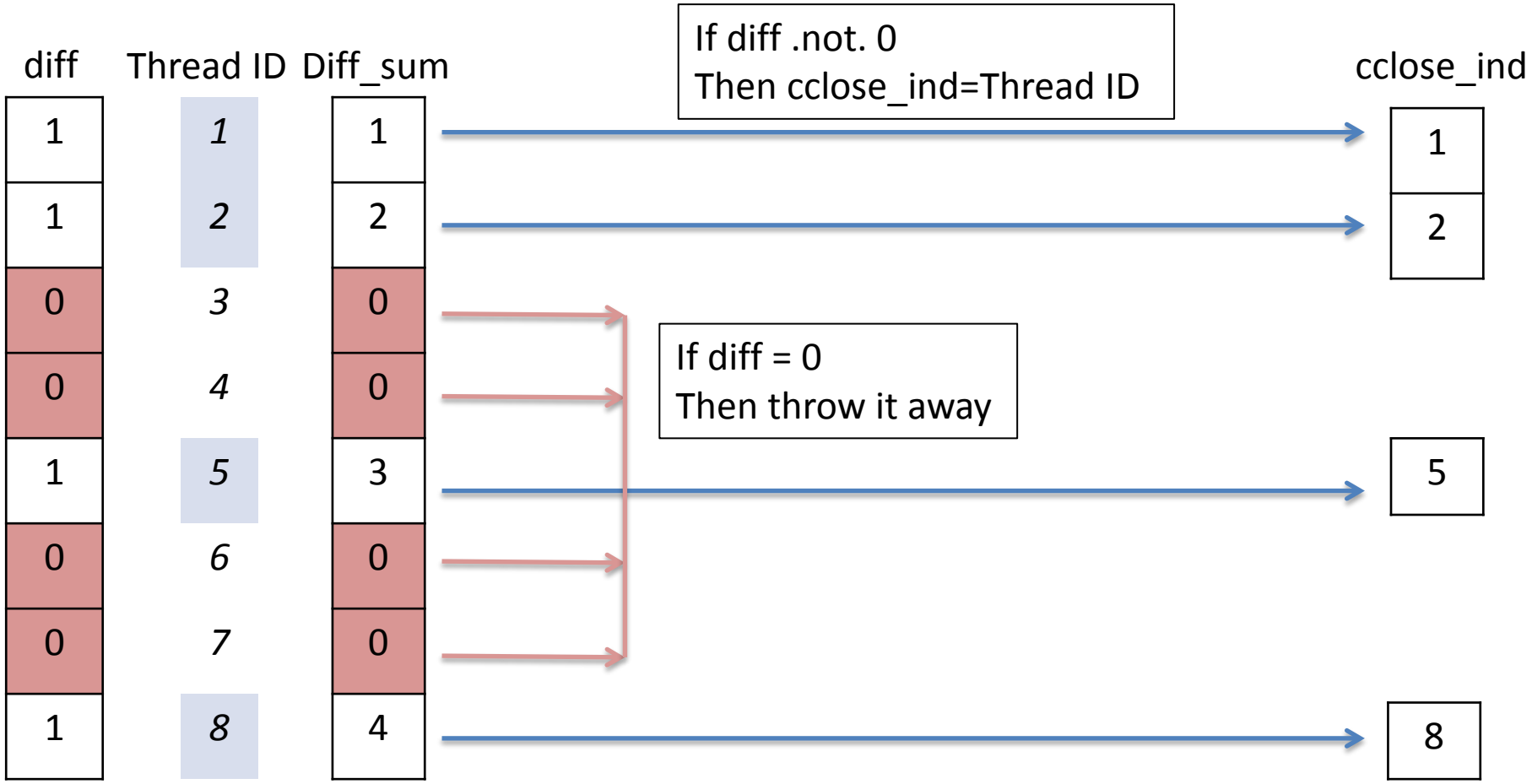
GPU Scan:

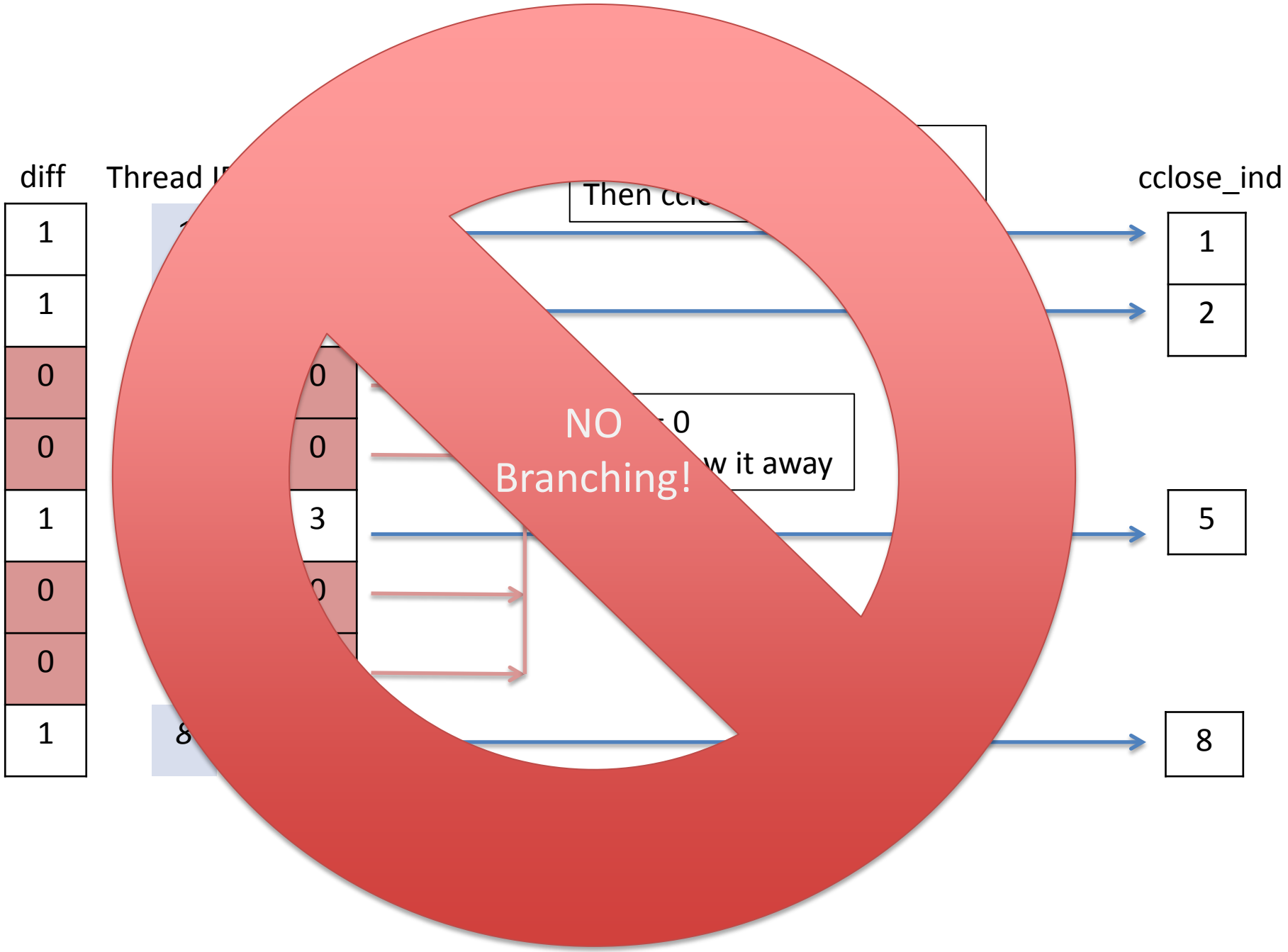

diff
1
1
0
0
1
0
0
1
dist
d1
d2
d3
d4
d5
d6
d7
d8
cdist
d1
d2
0
0
d5
0
0
d8
diff
1
1
0
0
1
0
0
1
psum
1
2
2
2
3
3
3
4
Diff_sum
1
2
0
0
3
0
0
4

Extract:

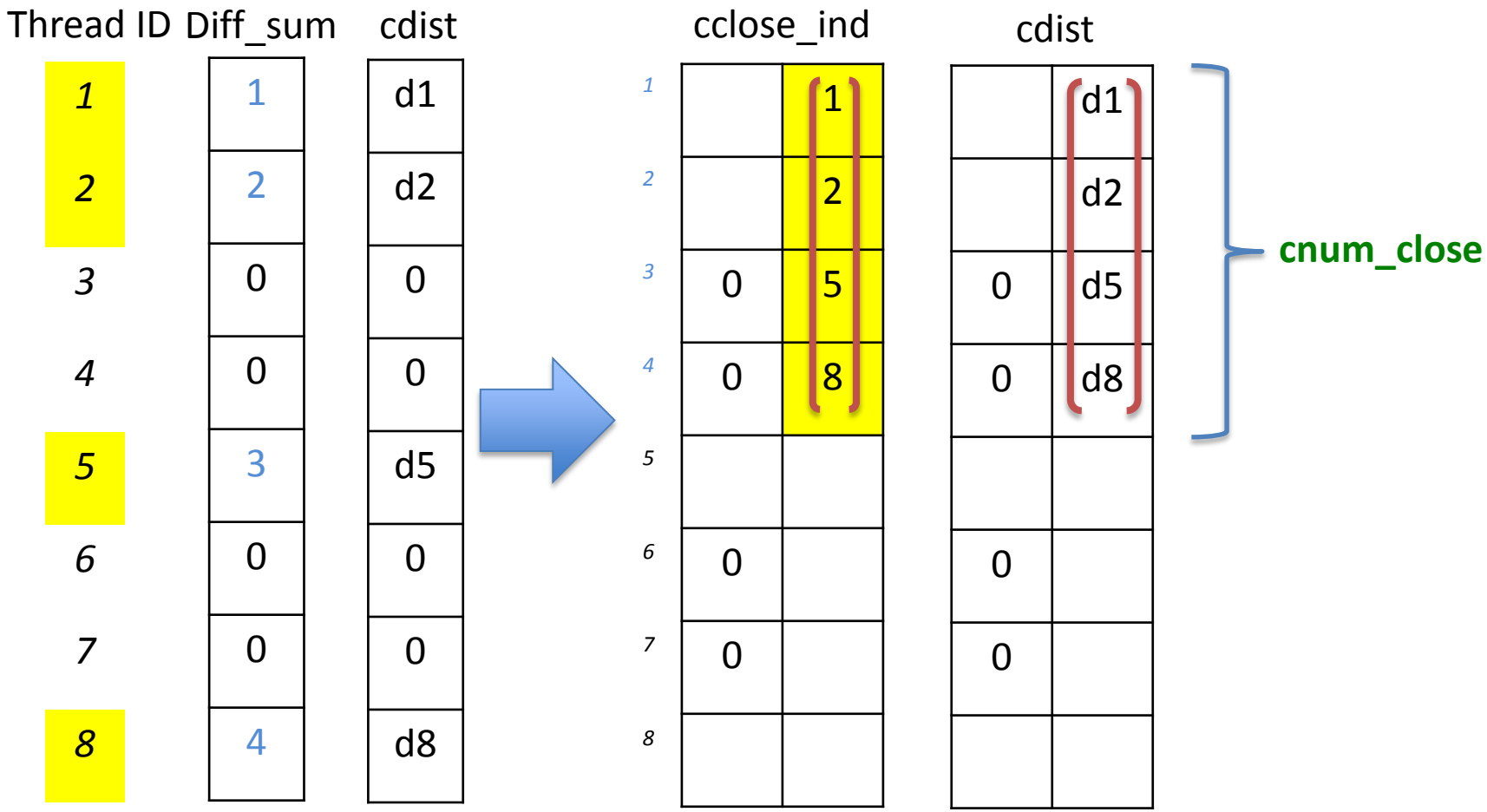

Thread ID
Diff_sum
cdist
1
2
3
4
5
6
7
8
1
2
0
0
3
0
0
4
d1
d2
0
0
d5
0
0
d8
What we want
What we have
cclose_ind
cdist
1
2
5
8
d1
d2
d5
d8
How can we independently
eliminate the zeros
and extract the indices
?

Solution?


diff
Thread ID
Diff_sum
1
1
0
0
1
0
0
1
1
2
3
4
5
6
7
8
1
2
0
0
3
0
0
4
If diff .not. 0
Then cclose_ind=Thread ID
If diff = 0
Then throw it away
cclose_ind
1
2
5
8


diff
Thread I
cclose_ind
1
1
0
0
1
0
0
1
0
0
3
0
1
2
5
8
NO
Branching!

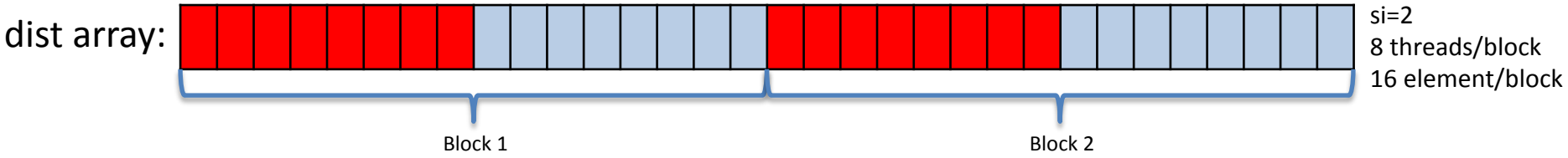
Solution!


Thread ID
Diff_sum
cdist
1
2
3
4
5
6
7
8
1
2
0
0
3
0
0
4
d1
d2
0
0
d5
0
0
d8
cclose_ind
cdist
1
2
3
4
5
6
7
8
1
2
5
8
0
0
0
0
d1
d2
d5
d8
0
0
0
0
cnum_close

Device Functions:

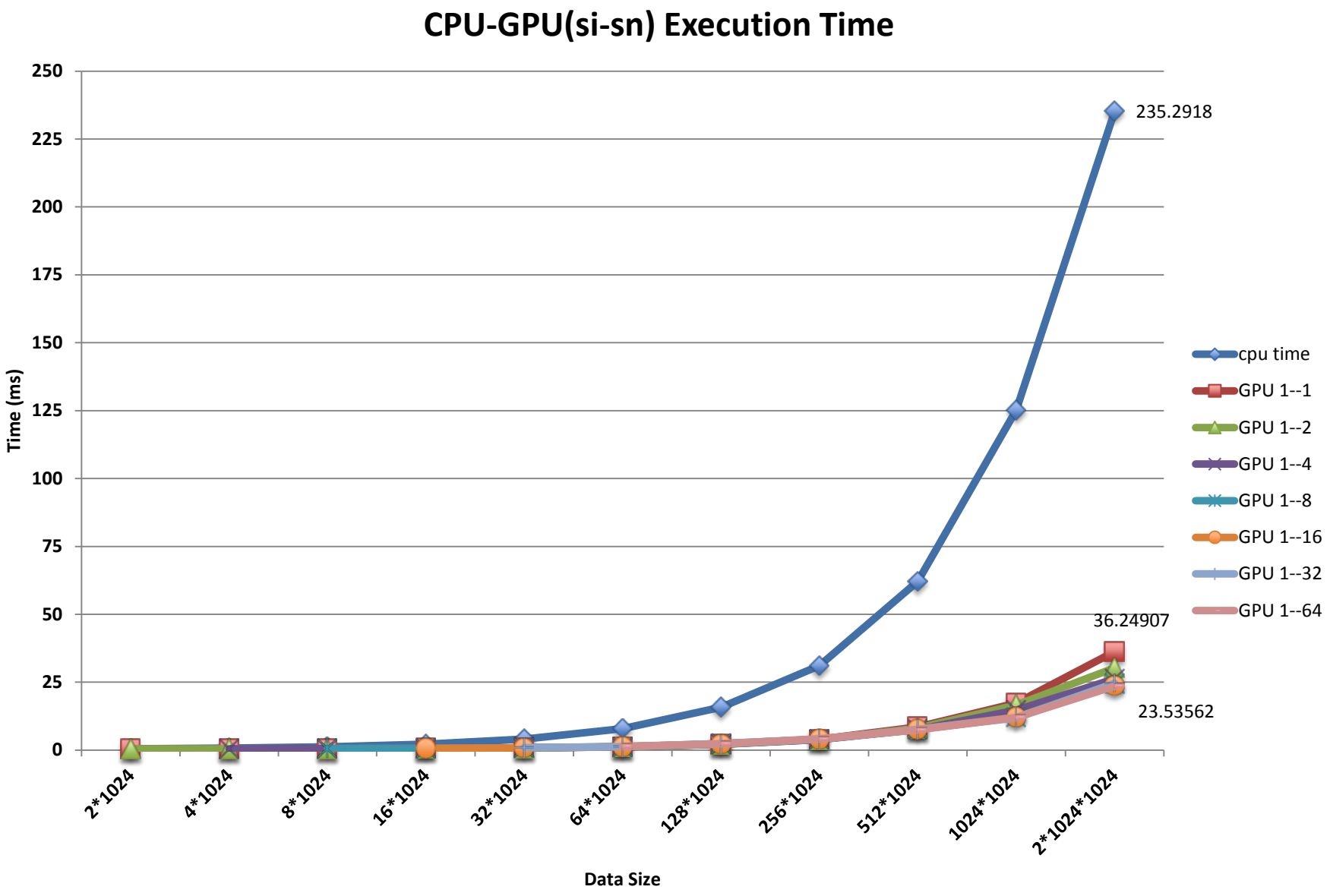
- gpu_dist
- gpu_scan

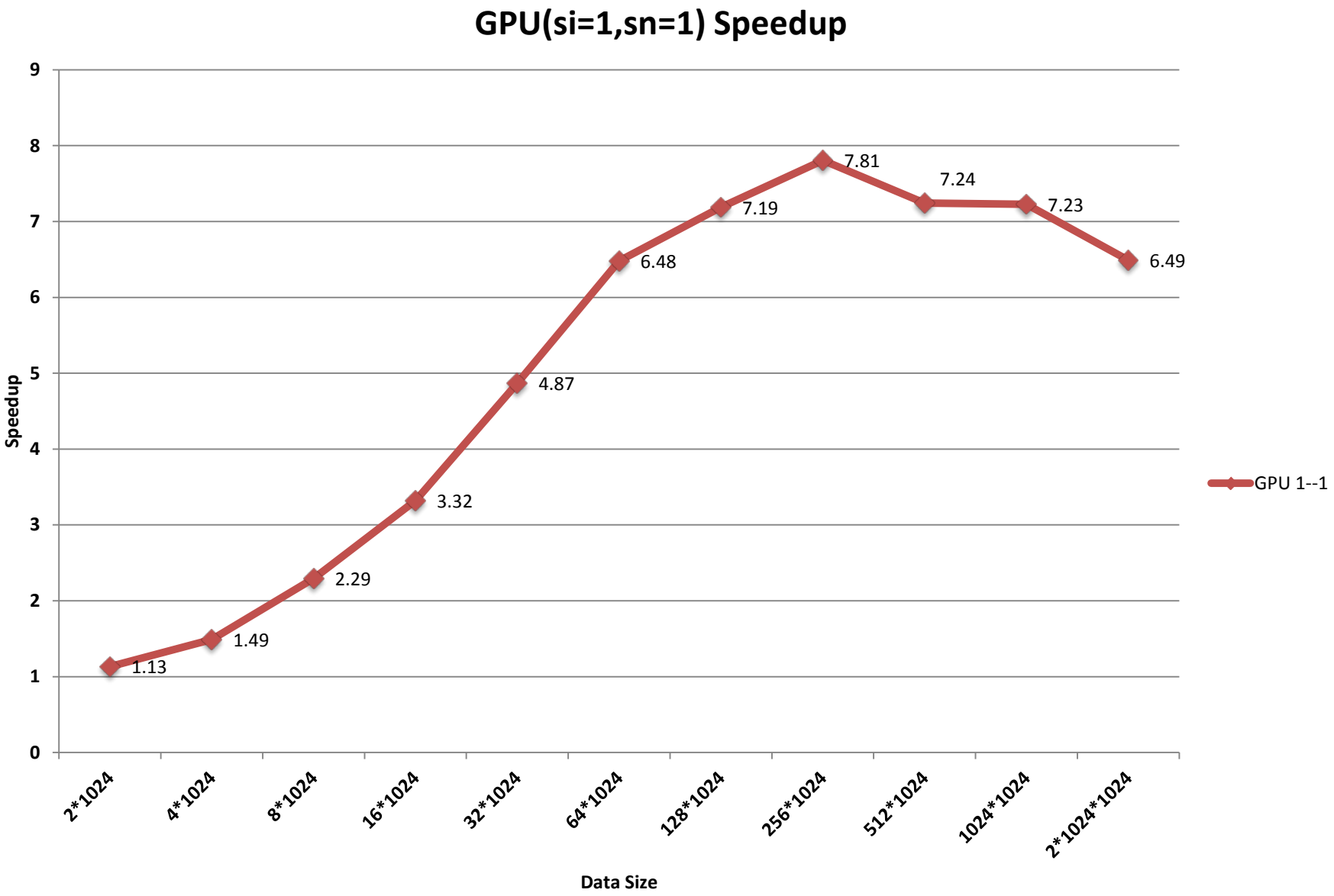
  si: number of iterations performed in this kernel.

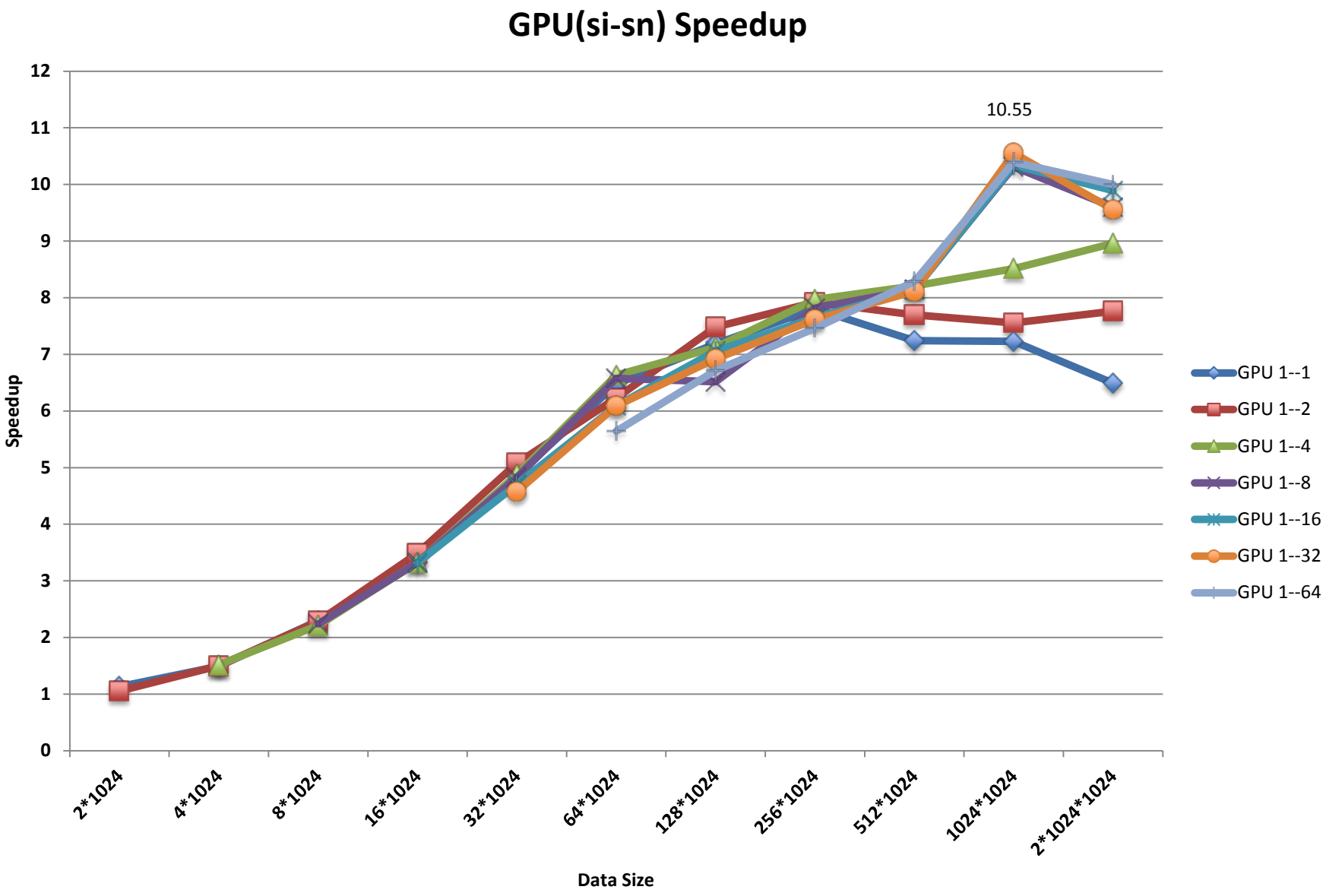


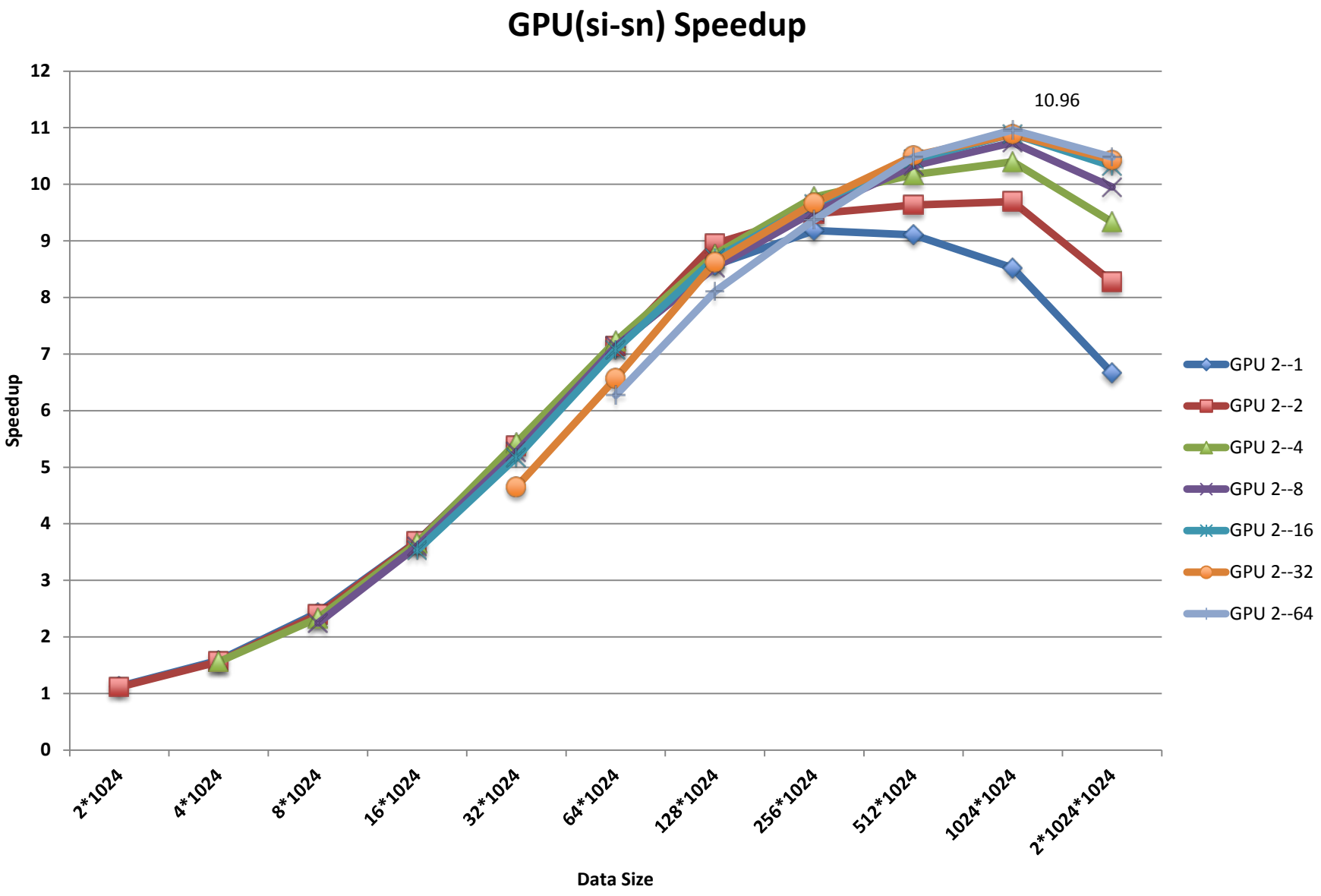
- extract

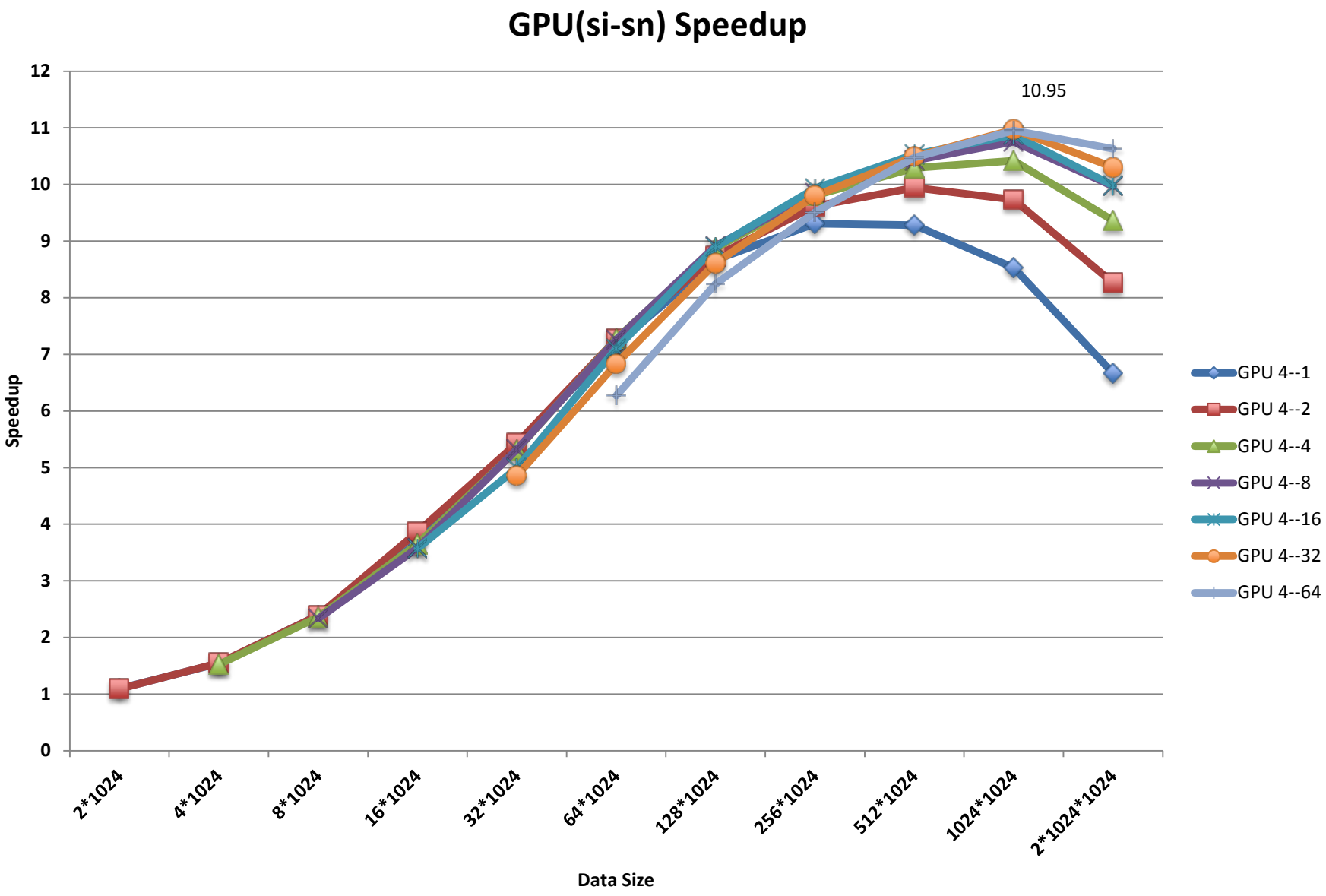
  sn: number of gpu_scan blocks that each extract block in this kernel handles.


CPU-GPU(si-sn) Execution Time
Time (ms)
250
225
200
175
150
125
100
75
50
25
0
235.2918
36.24907
23.53562
2*1024
4*1024
8*1024
16*1024
32*1024
64*1024
128*1024
256*1024
512*1024
1024*1024
2*1024*1024
Data Size
cpu time
GPU 1--1
GPU 1--2
GPU 1--4
GPU 1--8
GPU 1--16
GPU 1--32
GPU 1--64


GPU(si=1,sn=1) Speedup
Speedup
0
1
2
3
4
5
6
7
8
9
1.13
1.49
2.29
3.32
4.87
6.48
7.19
7.81
7.24
7.23
6.49
2*1024
4*1024
8*1024
16*1024
32*1024
64*1024
128*1024
256*1024
512*1024
1024*1024
2*1024*1024
Data Size
GPU 1--1


GPU(si-sn) Speedup
Speedup
Data Size
10.55
0
1
2
3
4
5
6
7
8
9
10
11
12
2*1024
4*1024
8*1024
16*1024
32*1024
64*1024
128*1024
256*1024
512*1024
1024*1024
2*1024*1024
GPU 1--1
GPU 1--2
GPU 1--4
GPU 1--8
GPU 1--16
GPU 1--32
GPU 1--64


GPU(si-sn) Speedup
10.96
Speedup
Data Size
0
1
2
3
4
5
6
7
8
9
10
11
12
2*1024
4*1024
8*1024
16*1024
32*1024
64*1024
128*1024
256*1024
512*1024
1024*1024
2*1024*1024
GPU 2--1
GPU 2--2
GPU 2--4
GPU 2--8
GPU 2--16
GPU 2--32
GPU 2--64


GPU(si-sn) Speedup
10.95
Speedup
Data Size
0
1
2
3
4
5
6
7
8
9
10
11
12
2*1024
4*1024
8*1024
16*1024
32*1024
64*1024
128*1024
256*1024
512*1024
1024*1024
2*1024*1024
GPU 4--1
GPU 4--2
GPU 4--4
GPU 4--8
GPU 4--16
GPU 4--32
GPU 4--64

Conclusion

- CUDA Fortran on GPU gave significant speedup vs CPU ($10x^{+}$).
- Step outside the box (redesign the algorithm).
- In order to get good performance, si and sn need to be tuned.
- Be careful with using device memory.
- There’s still room to improve the performance of this project.

Acknowledgements

DAReS/IMAGe

Helen Kershaw (Mentor)
Nancy Collins (Mentor)
Jeff Anderson
Tim Hoar
Kevin Raeder

Richard Loft
Raghu Raj Prasanna Kumar

Kristin Mooney
Silvia Gentile
Carolyn Mueller

UCAR **NCAR** **University of Wyoming**