

# 南开大学

## 计算机体系结构挑战赛报告



学院 计算机学院  
专业 计算机科学与技术  
南开大学 程伟卿  
学号 2311865

## 1 参赛选手信息

姓名	邮箱	电话
程伟卿	3075998327@qq.com	18860810166

注：我是单人组队。

## 2 比赛内容

近年来,随着人工智能和高性能计算的快速发展,GPU 编程已成为加速计算的重要手段。AMD ROCm 平台和 HIP 编程模型为开发者提供了强大的异构计算能力,使得复杂算法能够在 GPU 上高效执行。为了推动 GPU 编程技术的发展和应用,激发学生对并行计算的兴趣,本次比赛特设置三个具有代表性的 GPU 编程挑战题目。本次比赛要求参赛队伍使用 AMD ROCm 开源堆栈和 HIP 编程模型,在指定的 GPU 硬件平台上完成三个核心算法的高性能实现:

### 2.1 Prefix Sum (前缀和)

前缀和是并行计算中的基础算法之一,广泛应用于数据处理、图像处理和科学计算等领域。参赛者需要实现一个高效的 GPU 加速程序,计算包含数百万甚至数亿个整数的数组的前缀和。该算法要求对 GPU 的并行计算模式有深入理解,并能充分利用 GPU 的计算资源。

### 2.2 Softmax

Softmax 函数是深度学习和机器学习中的核心算法,常用于多分类问题的概率计算。参赛者需要实现一个数值稳定的 GPU Softmax 算法,能够处理大规模浮点数组。该题目考查参赛者对数值计算精度、GPU 内存管理和并行归约算法的掌握程度。

### 2.3 All-Pairs Shortest Path (APSP)

全源最短路径 APSP 是图算法中的经典问题,在网络分析、交通规划和社交网络分析等领域有重要应用。参赛者需要自主选择并实现任意一种 APSP 算法(如 Floyd-Warshall、Johnson 算法等),在 GPU 上高效求解有向加权图中任意两点间的最短路径。该题目考查参赛者的算法设计能力和 GPU 并行编程的综合应用能力。

## 3 配置信息

### 3.1 硬件平台

- GPU 型号: AMD Instinct MI100
- 计算环境: GPU 集群

### 3.2 软件要求

- 使用 AMD ROCm 开源堆栈和 HIP 编程模型
- 仅允许单 GPU 实现(不允许多 GPU)
- 必须自主实现算法核心逻辑(不得使用现成的高性能计算库)

- 代码必须在指定的 GPU 硬件平台上正常编译和运行

## 4 问题一：前缀和

### 4.1 基础框架与解题思路

常规解法直接顺序计算，修改 kernel.hip：

```
1 extern "C" void solve(const int* input, int* output, int N) {
2     output[0] = input[0];
3     for (int i = 1; i < N; i++) {
4         output[i] = output[i - 1] + input[i];
5     }
6 }
```

时间复杂度为  $O(n)$ 。

### 4.2 优化策略

#### 4.2.1 思路

本题要求在 GPU 上实现前缀和 (Prefix Sum)。串行算法为：

$$\text{output}[i] = \sum_{j=0}^i \text{input}[j], \quad i = 0, 1, \dots, N-1$$

其时间复杂度为  $O(N)$ ，但无法充分利用 GPU 并行能力。为此，我们采用 **Blelloch 扫描 (Scan) 算法**，其主要分为两个阶段：

- **上升阶段 (Upsweep / Reduce Phase)**：通过树形规约构建部分和；
- **下降阶段 (Downsweep Phase)**：反向传播前缀和，得到最终结果。

该算法在  $O(\log N)$  的并行步数内完成前缀和计算，并行度高，适合 GPU 实现。

#### 伪代码 (Blelloch 扫描)

```
1 Input: A[0..N-1]
2 Output: P[0..N-1] // prefix sums
3
4 // Phase 1: Upsweep
5 for d = 0 to log2(N)-1:
6     parallel for k = 0 to N-1:
7         if (k % 2^(d+1) == 2^(d+1)-1):
8             A[k] = A[k] + A[k - 2^d]
9
10 // Phase 2: Downsweep
11 A[N-1] = 0
12 for d = log2(N)-1 downto 0:
13     parallel for k = 0 to N-1:
14         if (k % 2^(d+1) == 2^(d+1)-1):
15             t = A[k - 2^d]
16             A[k - 2^d] = A[k]
17             A[k] = A[k] + t
```

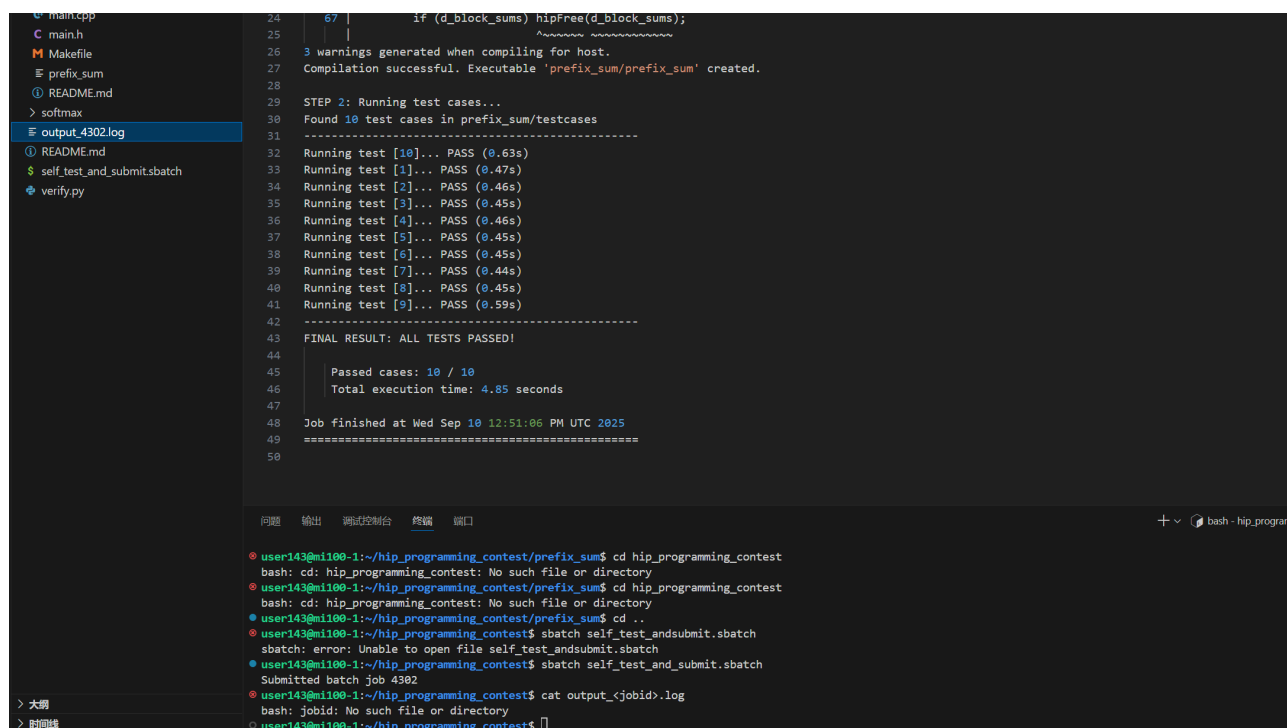
```
18
19 return P = A
```

说明 该算法具有以下特点：

1. 时间复杂度为  $O(\log N)$ ，空间复杂度为  $O(N)$ ；
2. 通过在共享内存中存储中间结果，减少了全局内存访问开销；
3. 适合在 HIP 中用线程块实现，每个 block 处理一段数据，再通过多 block 扫描完成全局前缀和。

## 4.3 优化结果

如下图，执行时间大幅缩短：



```
24 | 67 | if (d_block_sums) hipFree(d_block_sums);
25 |
26 | 3 warnings generated when compiling for host.
27 | Compilation successful. Executable 'prefix_sum/prefix_sum' created.
28 |
29 | STEP 2: Running test cases...
30 | Found 10 test cases in prefix_sum/testcases
31 | -----
32 | Running test [10]... PASS (0.63s)
33 | Running test [1]... PASS (0.47s)
34 | Running test [2]... PASS (0.46s)
35 | Running test [3]... PASS (0.45s)
36 | Running test [4]... PASS (0.46s)
37 | Running test [5]... PASS (0.45s)
38 | Running test [6]... PASS (0.45s)
39 | Running test [7]... PASS (0.44s)
40 | Running test [8]... PASS (0.45s)
41 | Running test [9]... PASS (0.59s)
42 | -----
43 | FINAL RESULT: ALL TESTS PASSED!
44 |
45 | Passed cases: 10 / 10
46 | Total execution time: 4.85 seconds
47 |
48 | Job finished at Wed Sep 10 12:51:06 PM UTC 2025
49 | =====
50 |
51 |
52 |
53 |
54 |
55 |
56 |
57 |
58 |
59 |
60 |
61 |
62 |
63 |
64 |
65 |
66 |
67 |
68 |
69 |
70 |
71 |
72 |
73 |
74 |
75 |
76 |
77 |
78 |
79 |
80 |
81 |
82 |
83 |
84 |
85 |
86 |
87 |
88 |
89 |
90 |
91 |
92 |
93 |
94 |
95 |
96 |
97 |
98 |
99 |
100 |
101 |
102 |
103 |
104 |
105 |
106 |
107 |
108 |
109 |
110 |
111 |
112 |
113 |
114 |
115 |
116 |
117 |
118 |
119 |
120 |
121 |
122 |
123 |
124 |
125 |
126 |
127 |
128 |
129 |
130 |
131 |
132 |
133 |
134 |
135 |
136 |
137 |
138 |
139 |
140 |
141 |
142 |
143 |
144 |
145 |
146 |
147 |
148 |
149 |
150 |
151 |
152 |
153 |
154 |
155 |
156 |
157 |
158 |
159 |
160 |
161 |
162 |
163 |
164 |
165 |
166 |
167 |
168 |
169 |
170 |
171 |
172 |
173 |
174 |
175 |
176 |
177 |
178 |
179 |
180 |
181 |
182 |
183 |
184 |
185 |
186 |
187 |
188 |
189 |
190 |
191 |
192 |
193 |
194 |
195 |
196 |
197 |
198 |
199 |
200 |
201 |
202 |
203 |
204 |
205 |
206 |
207 |
208 |
209 |
210 |
211 |
212 |
213 |
214 |
215 |
216 |
217 |
218 |
219 |
220 |
221 |
222 |
223 |
224 |
225 |
226 |
227 |
228 |
229 |
230 |
231 |
232 |
233 |
234 |
235 |
236 |
237 |
238 |
239 |
240 |
241 |
242 |
243 |
244 |
245 |
246 |
247 |
248 |
249 |
250 |
251 |
252 |
253 |
254 |
255 |
256 |
257 |
258 |
259 |
260 |
261 |
262 |
263 |
264 |
265 |
266 |
267 |
268 |
269 |
270 |
271 |
272 |
273 |
274 |
275 |
276 |
277 |
278 |
279 |
280 |
281 |
282 |
283 |
284 |
285 |
286 |
287 |
288 |
289 |
290 |
291 |
292 |
293 |
294 |
295 |
296 |
297 |
298 |
299 |
300 |
301 |
302 |
303 |
304 |
305 |
306 |
307 |
308 |
309 |
310 |
311 |
312 |
313 |
314 |
315 |
316 |
317 |
318 |
319 |
320 |
321 |
322 |
323 |
324 |
325 |
326 |
327 |
328 |
329 |
330 |
331 |
332 |
333 |
334 |
335 |
336 |
337 |
338 |
339 |
340 |
341 |
342 |
343 |
344 |
345 |
346 |
347 |
348 |
349 |
350 |
351 |
352 |
353 |
354 |
355 |
356 |
357 |
358 |
359 |
360 |
361 |
362 |
363 |
364 |
365 |
366 |
367 |
368 |
369 |
370 |
371 |
372 |
373 |
374 |
375 |
376 |
377 |
378 |
379 |
380 |
381 |
382 |
383 |
384 |
385 |
386 |
387 |
388 |
389 |
390 |
391 |
392 |
393 |
394 |
395 |
396 |
397 |
398 |
399 |
400 |
401 |
402 |
403 |
404 |
405 |
406 |
407 |
408 |
409 |
410 |
411 |
412 |
413 |
414 |
415 |
416 |
417 |
418 |
419 |
420 |
421 |
422 |
423 |
424 |
425 |
426 |
427 |
428 |
429 |
430 |
431 |
432 |
433 |
434 |
435 |
436 |
437 |
438 |
439 |
440 |
441 |
442 |
443 |
444 |
445 |
446 |
447 |
448 |
449 |
450 |
451 |
452 |
453 |
454 |
455 |
456 |
457 |
458 |
459 |
460 |
461 |
462 |
463 |
464 |
465 |
466 |
467 |
468 |
469 |
470 |
471 |
472 |
473 |
474 |
475 |
476 |
477 |
478 |
479 |
480 |
481 |
482 |
483 |
484 |
485 |
486 |
487 |
488 |
489 |
490 |
491 |
492 |
493 |
494 |
495 |
496 |
497 |
498 |
499 |
500 |
501 |
502 |
503 |
504 |
505 |
506 |
507 |
508 |
509 |
510 |
511 |
512 |
513 |
514 |
515 |
516 |
517 |
518 |
519 |
520 |
521 |
522 |
523 |
524 |
525 |
526 |
527 |
528 |
529 |
530 |
531 |
532 |
533 |
534 |
535 |
536 |
537 |
538 |
539 |
540 |
541 |
542 |
543 |
544 |
545 |
546 |
547 |
548 |
549 |
550 |
551 |
552 |
553 |
554 |
555 |
556 |
557 |
558 |
559 |
560 |
561 |
562 |
563 |
564 |
565 |
566 |
567 |
568 |
569 |
570 |
571 |
572 |
573 |
574 |
575 |
576 |
577 |
578 |
579 |
580 |
581 |
582 |
583 |
584 |
585 |
586 |
587 |
588 |
589 |
590 |
591 |
592 |
593 |
594 |
595 |
596 |
597 |
598 |
599 |
600 |
601 |
602 |
603 |
604 |
605 |
606 |
607 |
608 |
609 |
610 |
611 |
612 |
613 |
614 |
615 |
616 |
617 |
618 |
619 |
620 |
621 |
622 |
623 |
624 |
625 |
626 |
627 |
628 |
629 |
630 |
631 |
632 |
633 |
634 |
635 |
636 |
637 |
638 |
639 |
640 |
641 |
642 |
643 |
644 |
645 |
646 |
647 |
648 |
649 |
650 |
651 |
652 |
653 |
654 |
655 |
656 |
657 |
658 |
659 |
660 |
661 |
662 |
663 |
664 |
665 |
666 |
667 |
668 |
669 |
670 |
671 |
672 |
673 |
674 |
675 |
676 |
677 |
678 |
679 |
680 |
681 |
682 |
683 |
684 |
685 |
686 |
687 |
688 |
689 |
690 |
691 |
692 |
693 |
694 |
695 |
696 |
697 |
698 |
699 |
700 |
701 |
702 |
703 |
704 |
705 |
706 |
707 |
708 |
709 |
710 |
711 |
712 |
713 |
714 |
715 |
716 |
717 |
718 |
719 |
720 |
721 |
722 |
723 |
724 |
725 |
726 |
727 |
728 |
729 |
730 |
731 |
732 |
733 |
734 |
735 |
736 |
737 |
738 |
739 |
740 |
741 |
742 |
743 |
744 |
745 |
746 |
747 |
748 |
749 |
750 |
751 |
752 |
753 |
754 |
755 |
756 |
757 |
758 |
759 |
760 |
761 |
762 |
763 |
764 |
765 |
766 |
767 |
768 |
769 |
770 |
771 |
772 |
773 |
774 |
775 |
776 |
777 |
778 |
779 |
780 |
781 |
782 |
783 |
784 |
785 |
786 |
787 |
788 |
789 |
790 |
791 |
792 |
793 |
794 |
795 |
796 |
797 |
798 |
799 |
800 |
801 |
802 |
803 |
804 |
805 |
806 |
807 |
808 |
809 |
810 |
811 |
812 |
813 |
814 |
815 |
816 |
817 |
818 |
819 |
820 |
821 |
822 |
823 |
824 |
825 |
826 |
827 |
828 |
829 |
830 |
831 |
832 |
833 |
834 |
835 |
836 |
837 |
838 |
839 |
840 |
841 |
842 |
843 |
844 |
845 |
846 |
847 |
848 |
849 |
850 |
851 |
852 |
853 |
854 |
855 |
856 |
857 |
858 |
859 |
860 |
861 |
862 |
863 |
864 |
865 |
866 |
867 |
868 |
869 |
870 |
871 |
872 |
873 |
874 |
875 |
876 |
877 |
878 |
879 |
880 |
881 |
882 |
883 |
884 |
885 |
886 |
887 |
888 |
889 |
890 |
891 |
892 |
893 |
894 |
895 |
896 |
897 |
898 |
899 |
900 |
901 |
902 |
903 |
904 |
905 |
906 |
907 |
908 |
909 |
910 |
911 |
912 |
913 |
914 |
915 |
916 |
917 |
918 |
919 |
920 |
921 |
922 |
923 |
924 |
925 |
926 |
927 |
928 |
929 |
930 |
931 |
932 |
933 |
934 |
935 |
936 |
937 |
938 |
939 |
940 |
941 |
942 |
943 |
944 |
945 |
946 |
947 |
948 |
949 |
950 |
951 |
952 |
953 |
954 |
955 |
956 |
957 |
958 |
959 |
960 |
961 |
962 |
963 |
964 |
965 |
966 |
967 |
968 |
969 |
970 |
971 |
972 |
973 |
974 |
975 |
976 |
977 |
978 |
979 |
980 |
981 |
982 |
983 |
984 |
985 |
986 |
987 |
988 |
989 |
990 |
991 |
992 |
993 |
994 |
995 |
996 |
997 |
998 |
999 |
1000 |
1001 |
1002 |
1003 |
1004 |
1005 |
1006 |
1007 |
1008 |
1009 |
1010 |
1011 |
1012 |
1013 |
1014 |
1015 |
1016 |
1017 |
1018 |
1019 |
1020 |
1021 |
1022 |
1023 |
1024 |
1025 |
1026 |
1027 |
1028 |
1029 |
1030 |
1031 |
1032 |
1033 |
1034 |
1035 |
1036 |
1037 |
1038 |
1039 |
1040 |
1041 |
1042 |
1043 |
1044 |
1045 |
1046 |
1047 |
1048 |
1049 |
1050 |
1051 |
1052 |
1053 |
1054 |
1055 |
1056 |
1057 |
1058 |
1059 |
1060 |
1061 |
1062 |
1063 |
1064 |
1065 |
1066 |
1067 |
1068 |
1069 |
1070 |
1071 |
1072 |
1073 |
1074 |
1075 |
1076 |
1077 |
1078 |
1079 |
1080 |
1081 |
1082 |
1083 |
1084 |
1085 |
1086 |
1087 |
1088 |
1089 |
1090 |
1091 |
1092 |
1093 |
1094 |
1095 |
1096 |
1097 |
1098 |
1099 |
1100 |
1101 |
1102 |
1103 |
1104 |
1105 |
1106 |
1107 |
1108 |
1109 |
1110 |
1111 |
1112 |
1113 |
1114 |
1115 |
1116 |
1117 |
1118 |
1119 |
1120 |
1121 |
1122 |
1123 |
1124 |
1125 |
1126 |
1127 |
1128 |
1129 |
1130 |
1131 |
1132 |
1133 |
1134 |
1135 |
1136 |
1137 |
1138 |
1139 |
1140 |
1141 |
1142 |
1143 |
1144 |
1145 |
1146 |
1147 |
1148 |
1149 |
1150 |
1151 |
1152 |
1153 |
1154 |
1155 |
1156 |
1157 |
1158 |
1159 |
1160 |
1161 |
1162 |
1163 |
1164 |
1165 |
1166 |
1167 |
1168 |
1169 |
1170 |
1171 |
1172 |
1173 |
1174 |
1175 |
1176 |
1177 |
1178 |
1179 |
1180 |
1181 |
1182 |
1183 |
1184 |
1185 |
1186 |
1187 |
1188 |
1189 |
1190 |
1191 |
1192 |
1193 |
1194 |
1195 |
1196 |
1197 |
1198 |
1199 |
1200 |
1201 |
1202 |
1203 |
1204 |
1205 |
1206 |
1207 |
1208 |
1209 |
1210 |
1211 |
1212 |
1213 |
1214 |
1215 |
1216 |
1217 |
1218 |
1219 |
1220 |
1221 |
1222 |
1223 |
1224 |
1225 |
1226 |
1227 |
1228 |
1229 |
1230 |
1231 |
1232 |
1233 |
1234 |
1235 |
1236 |
1237 |
1238 |
1239 |
1240 |
1241 |
1242 |
1243 |
1244 |
1245 |
1246 |
1247 |
1248 |
1249 |
1250 |
1251 |
1252 |
1253 |
1254 |
1255 |
1256 |
1257 |
1258 |
1259 |
1260 |
1261 |
1262 |
1263 |
1264 |
1265 |
1266 |
1267 |
1268 |
1269 |
1270 |
1271 |
1272 |
1273 |
1274 |
1275 |
1276 |
1277 |
1278 |
1279 |
1280 |
1281 |
1282 |
1283 |
1284 |
1285 |
1286 |
1287 |
1288 |
1289 |
1290 |
1291 |
1292 |
1293 |
1294 |
1295 |
1296 |
1297 |
1298 |
1299 |
1300 |
1301 |
1302 |
1303 |
1304 |
1305 |
1306 |
1307 |
1308 |
1309 |
1310 |
1311 |
1312 |
1313 |
1314 |
1315 |
1316 |
1317 |
1318 |
1319 |
1320 |
1321 |
1322 |
1323 |
1324 |
1325 |
1326 |
1327 |
1328 |
1329 |
1330 |
1331 |
1332 |
1333 |
1334 |
1335 |
1336 |
1337 |
1338 |
1339 |
1340 |
1341 |
1342 |
1343 |
1344 |
1345 |
1346 |
1347 |
1348 |
1349 |
1350 |
1351 |
1352 |
1353 |
1354 |
1355 |
1356 |
1357 |
1358 |
1359 |
1360 |
1361 |
1362 |
1363 |
1364 |
1365 |
1366 |
1367 |
1368 |
1369 |
1370 |
1371 |
1372 |
1373 |
1374 |
1375 |
1376 |
1377 |
1378 |
1379 |
1380 |
1381 |
1382 |
1383 |
1384 |
1385 |
1386 |
1387 |
1388 |
1389 |
1390 |
1391 |
1392 |
1393 |
1394 |
1395 |
1396 |
1397 |
1398 |
1399 |
1400 |
1401 |
1402 |
1403 |
1404 |
1405 |
1406 |
1407 |
1408 |
1409 |
1410 |
1411 |
1412 |
1413 |
1414 |
1415 |
1416 |
1417 |
1418 |
1419 |
1420 |
1421 |
1422 |
1423 |
1424 |
1425 |
1426 |
1427 |
1428 |
1429 |
1430 |
1431 |
1432 |
1433 |
1434 |
1435 |
1436 |
1437 |
1438 |
1439 |
1440 |
1441 |
1442 |
1443 |
1444 |
1445 |
1446 |
1447 |
1448 |
1449 |
1450 |
1451 |
1452 |
1453 |
1454 |
1455 |
1456 |
1457 |
1458 |
1459 |
1460 |
1461 |
1462 |
1463 |
1464 |
1465 |
1466 |
1467 |
1468 |
1469 |
1470 |
1471 |
1472 |
1473 |
1474 |
1475 |
1476 |
1477 |
1478 |
1479 |
1480 |
1481 |
1482 |
1483 |
1484 |
1485 |
1486 |
1487 |
1488 |
1489 |
1490 |
1491 |
1492 |
1493 |
1494 |
1495 |
1496 |
1497 |
1498 |
1499 |
1500 |
1501 |
1502 |
1503 |
1504 |
1505 |
1506 |
1507 |
1508 |
1509 |
1510 |
1511 |
1512 |
1513 |
1514 |
1515 |
1516 |
1517 |
1518 |
1519 |
1520 |
1521 |
1522 |
1523 |
1524 |
1525 |
1526 |
1527 |
1528 |
1529 |
1530 |
1531 |
1532 |
1533 |
1534 |
1535 |
1536 |
1537 |
1538 |
1539 |
1540 |
1541 |
1542 |
1543 |
1544 |
1545 |
1546 |
1547 |
1548 |
1549 |
1550 |
1551 |
1552 |
1553 |
1554 |
1555 |
1556 |
1557 |
1558 |
1559 |
1560 |
1561 |
1562 |
1563 |
1564 |
1565 |
1566 |
1567 |
1568 |
1569 |
1570 |
1571 |
1572 |
1573 |
1574 |
1575 |
1576 |
1577 |
1578 |
1579 |
1580 |
1581 |
1582 |
1583 |
1584 |
1585 |
1586 |
1587 |
1588 |
1589 |
1590 |
1591 |
1592 |
1593 |
1594 |
1595 |
1596 |
1597 |
1598 |
1599 |
1600 |
1601 |
1602 |
1603 |
1604 |
1605 |
1606 |
1607 |
1608 |
1609 |
1610 |
1611 |
1612 |
1613 |
1614 |
1615 |
1616 |
1617 |
1618 |
1619 |
1620 |
1621 |
1622 |
1623 |
1624 |
1625 |
1626 |
1627 |
1628 |
1629 |
1630 |
1631 |
1632 |
1633 |
1634 |
1635 |
1636 |
1637 |
1638 |
1639 |
1640 |
1641 |
1642 |
1643 |
1644 |
1645 |
1646 |
1647 |
1648 |
1649 |
1650 |
1651 |
1652 |
1653 |
1654 |
1655 |
1656 |
1657 |
1658 |
1659 |
1660 |
1661 |
1662 |
1663 |
1664 |
1665 |
1666 |
1667 |
1668 |
1669 |
1670 |
1671 |
1672 |
1673 |
1674 |
1675 |
1676 |
1677 |
1678 |
1679 |
1680 |
1681 |
1682 |
1683 |
1684 |
1685 |
1686 |
1687 |
1688 |
1689 |
1690 |
1691 |
1692 |
1693 |
1694 |
1695 |
1696 |
1697 |
1698 |
1699 |
1700 |
1701 |
1702 |
1703 |
1704 |
1705 |
1706 |
1707 |
1708 |
1709 |
1710 |
1711 |
1712 |
1713 |
1714 |
1715 |
1716 |
1717 |
1718 |
1719 |
1720 |
1721 |
1722 |
1723 |
1724 |
1725 |
1726 |
1727 |
1728 |
1729 |
1730 |
1731 |
1732 |
1733 |
1734 |
1735 |
1736 |
1737 |
1738 |
1739 |
1740 |
1741 |
1742 |
1743 |
1744 |
1745 |
1746 |
1747 |
1748 |
1749 |
1750 |
1751 |
1752 |
1753 |
1754 |
1755 |
1756 |
1757 |
1758 |
1759 |
1760 |
1761 |
1762 |
1763 |
1764 |
1765 |
1766 |
1767 |
1768 |
1769 |
1770 |
1771 |
1772 |
1773 |
1774 |
1775 |
1776 |
1777 |
1778 |
1779 |
1780 |
1781 |
1782 |
1783 |
1784 |
1785 |
1786 |
1787 |
1788 |
1789 |
1790 |
1791 |
1792 |
1793 |
1794 |
1795 |
1796 |
1797 |
1798 |
1799 |
1800 |
1801 |
1802 |
1803 |
1804 |
1805 |
1806 |
1807 |
1808 |
1809 |
1810 |
1811 |
1812 |
1813 |
1814 |
1815 |
1816 |
1817 |
1818 |
1819 |
1820 |
1821 |
1822 |
1823 |
1824 |
1825 |
1826 |
1827 |
1828 |
1829 |
1830 |
1831 |
1832 |
1833 |
1834 |
1835 |
1836 |
1837 |
1838 |
1839 |
1840 |
1841 |
1842 |
1843 |
1844 |
1845 |
1846 |
1847 |
1848 |
1849 |
1850 |
1851 |
1852 |
1853 |
1854 |
1855 |
1856 |
1857 |
1858 |
1859 |
1860 |
1861 |
1862 |
1863 |
1864 |
1865 |
1866 |
1867 |
1868 |
1869 |
1870 |
1871 |
1872 |
1873 |
1874 |
1875 |
1876 |
1877 |
1878 |
1879 |
1880 |
1881 |
1882 |
1883 |
1884 |
1885 |
1886 |
1887 |
1888 |
1889 |
1890 |
1891 |
1892 |
1893 |
1894 |
1895 |
1896 |
1897 |
1898 |
1899 |
1900 |
1901 |
1902 |
1903 |
1904 |
1905 |
1906 |
1907 |
1908 |
1909 |
1910 |
1911 |
1912 |
1913 |
1914 |
1915 |
1916 |
1917 |
1918 |
1919 |
1920 |
1921 |
1922 |
1923 |
1924 |
1925 |
1926 |
1927 |
1928 |
1929 |
1930 |
1931 |
1932 |
1933 |
1934 |
1935 |
1936 |
1937 |
1938 |
1939 |
1940 |
1941 |
1942 |
1943 |
1944 |
1945 |
1946 |
1947 |
1948 |
1949 |
1950 |
1951 |
1952 |
1953 |
1954 |
1955 |
1956 |
1957 |
1958 |
1959 |
1960 |
1961 |
1962 |
1963 |
1964 |
1965 |
1966 |
1967 |
1968 |
1969 |
1970 |
1971 |
1972 |
1973 |
1974 |
1975 |
1976 |
1977 |
1978 |
1979 |
1980 |
1981 |
1982 |
1983 |
1984 |
1985 |
1986 |
1987 |
1988 |
1989 |
1990 |
1991 |
1992 |
1993 |
1994 |
1995 |
1996 |
1997 |
1998 |
1999 |
2000 |
2001 |
2002 |
2003 |
2004 |
2005 |
2006 |
2007 |
2008 |
2009 |
2010 |
2011 |
2012 |
2013 |
2014 |
2015 |
2016 |
2017 |
2018 |
2019 |
2020 |
2021 |
2022 |
2023 |
2024 |
2025 |
2026 |
2027 |
2028 |
2029 |
2030 |
2031 |
2032 |
2033 |
2034 |
2035 |
2036 |
2037 |
2038 |
2039 |
2040 |
2041 |
2042 |
2043 |
2044 |
2045 |
2046 |
2047 |
2048 |
2049 |
2050 |
2051 |
2052 |
2053 |
2054 |
2055 |
2056 |
2057 |
2058 |
2059 |
2060 |
2061 |
2062 |
2063 |
2064 |
2065 |
2066 |
2067 |
2068 |
2069 |
2070 |
2071 |
2072 |
2073 |
2074 |
2075 |
2076 |
2077 |
2078 |
2079 |
2080 |
2081 |
2082 |
2083 |
2084 |
2085 |
2086 |
2087 |
2088 |
2089 |
2090 |
2091 |
2092 |
2093 |
2094 |
2095 |
2096 |
2097 |
2098 |
2099 |
2100 |
2101 |
2102 |
2103 |
2104 |
2105 |
2106 |
2107 |
2108 |
2109 |
2110 |
2111 |
2112 |
2113 |
2114 |
2115 |
2116 |
2117 |
2118 |
2119 |
2120 |
2121 |
2122 |
2123 |
2124 |
2125 |
2126 |
2127 |
2128 |
2129 |
2130 |
2131 |
2132 |
2133 |
2134 |
2135 |
2136 |
2137 |
2138 |
2139 |
2140 |
2141 |
2142 |
2143 |
2144 |
2145 |
2146 |
2147 |
2148 |
2149 |
2150 |
2151 |
2152 |
2153 |
2154 |
2155 |
```

表 1: Test Results Summary

Test Case	Result	Time (s)
1	PASS	0.47
2	PASS	0.46
3	PASS	0.45
4	PASS	0.46
5	PASS	0.45
6	PASS	0.45
7	PASS	0.44
8	PASS	0.45
9	PASS	0.59
10	PASS	0.63
<b>Total</b>	<b>10 / 10 PASS</b>	<b>4.85</b>

$$\text{softmax}(x_i) = \frac{e^{x_i - \max(x)}}{\sum_{j=1}^n e^{x_j - \max(x)}}$$

解题步骤如下：

- 读取输入
- 求最大值
- 计算指数和
- 归一化

时间复杂度为  $O(n)$ ，遍历数组三次。

伪代码如下：

---

```

1 function Softmax(input: array of float, N: integer) -> array of float
2     output = new array of size N
3
4     # 1. 找最大值, 保证数值稳定
5     max_val = input[0]
6     for i from 1 to N-1 do
7         if input[i] > max_val then
8             max_val = input[i]
9         end if
10    end for
11
12    # 2. 计算指数和
13    sum_exp = 0.0
14    for i from 0 to N-1 do
15        output[i] = exp(input[i] - max_val)
16        sum_exp = sum_exp + output[i]
17    end for
18
19    # 3. 归一化

```

```

20     for i from 0 to N-1 do
21         output[i] = output[i] / sum_exp
22     end for
23
24     return output
25 end function

```

---

## 5.2 优化策略

本节介绍 Softmax 算法在 GPU 上的高性能优化策略，旨在提升计算效率、保证数值稳定，并充分利用 AMD GPU 并行计算能力。

1. **数值稳定性** Softmax 的原始公式为：

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

当输入值较大或较小时，直接计算指数容易造成溢出或下溢。为解决该问题，我们采用数值稳定的优化：

$$\text{softmax}(x_i) = \frac{e^{x_i - \max(x)}}{\sum_j e^{x_j - \max(x)}}$$

即在计算指数前，将每行向量减去该行最大值。

2. **GPU 并行化设计** 为了充分利用 GPU 并行计算能力，Softmax 的计算过程被拆分为三个核心步骤：

1. **求最大值 (Max Reduction)**: 每个线程处理一个元素，利用共享内存进行块内归约，得到每个线程块的局部最大值，最终在主机上归约得到全局最大值。

```

// pseudo-code: block-wise max reduction
shared float sdata[BLOCK_SIZE];
idx = threadIdx + blockIdx * blockDim;
sdata[threadIdx] = input[idx];
__syncthreads();
for (int s = blockDim/2; s > 0; s >>= 1) {
    if (threadIdx < s)
        sdata[threadIdx] = max(sdata[threadIdx], sdata[threadIdx+s]);
    __syncthreads();
}
if (threadIdx == 0) block_max[blockIdx] = sdata[0];

```

2. **计算指数 (Exponentiation)**: 每个线程独立计算  $e^{x_i - \max(x)}$ ，保证并行计算的独立性和内存访问的连续性。

```

// pseudo-code: compute exponentials
idx = threadIdx + blockIdx * blockDim;
output[idx] = exp(input[idx] - max_val);

```

3. **求和并归一化 (Sum Reduction & Normalization)**: 类似最大值归约, 每个线程块计算部分和, 然后在主机上求总和, 最后在 GPU 上进行归一化:

$$y_i = \frac{e^{x_i - \max(x)}}{\sum_j e^{x_j - \max(x)}}$$

```
// pseudo-code: sum reduction and normalization
shared float sdata[BLOCK_SIZE];
sdata[threadIdx] = output[idx];
__syncthreads();
for (int s = blockDim/2; s > 0; s >>= 1)
    if (threadIdx < s)
        sdata[threadIdx] += sdata[threadIdx+s];
__syncthreads();
if (threadIdx == 0) block_sum[blockIdx] = sdata[0];

// normalization kernel
output[idx] /= sum_exp;
```

### 3. 内存优化

- 使用 **共享内存** 进行线程块内归约, 减少对全局内存的访问次数。
- 全局内存访问尽量连续, 确保 coalesced memory, 提高带宽利用率。
- 每个线程处理一个元素, 避免线程间依赖, 提升并行效率。

### 4. 参数调优

- **BLOCK\_SIZE**: 根据 GPU 架构选择适当的线程块大小 (通常 128–256)。
- **线程处理元素数量**: 对于大规模向量, 每个线程可处理多个元素 (loop unrolling) 以减少归约次数。

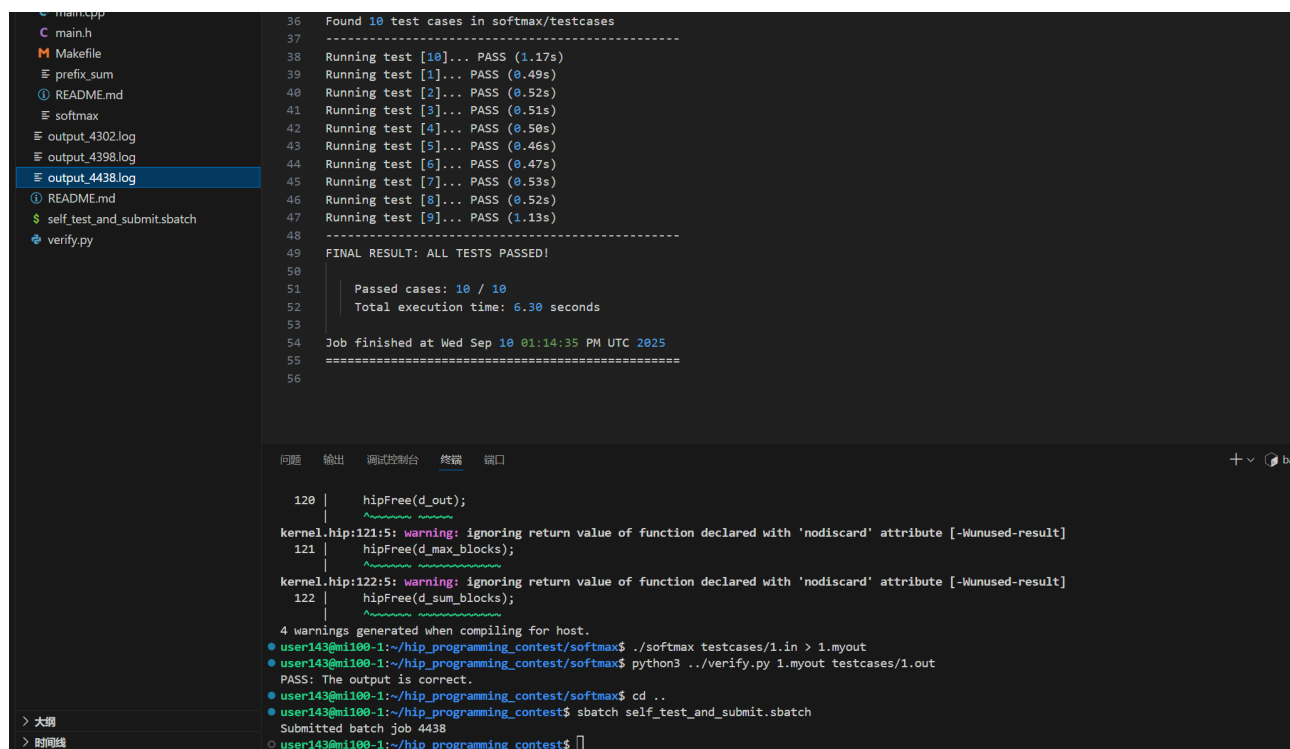
### 5. 总结 通过上述优化策略:

- 保证 Softmax 数值稳定。
- 充分利用 GPU 并行计算资源。
- 高效完成最大值归约、指数计算和归一化步骤。

此方法可在单 GPU 上对大规模向量实现高性能 Softmax 计算。

## 5.3 优化结果

优化后结果如下图：



```
36 Found 10 test cases in softmax/testcases
37 -----
38 Running test [10]... PASS (1.17s)
39 Running test [1]... PASS (0.49s)
40 Running test [2]... PASS (0.52s)
41 Running test [3]... PASS (0.51s)
42 Running test [4]... PASS (0.50s)
43 Running test [5]... PASS (0.46s)
44 Running test [6]... PASS (0.47s)
45 Running test [7]... PASS (0.53s)
46 Running test [8]... PASS (0.52s)
47 Running test [9]... PASS (1.13s)
48 -----
49 FINAL RESULT: ALL TESTS PASSED!
50
51 Passed cases: 10 / 10
52 Total execution time: 6.30 seconds
53
54 Job finished at Wed Sep 10 01:14:35 PM UTC 2025
55 =====
56
```

```
120 | hipFree(d_out);
    | ~~~~~
kernel.hip:121:5: warning: ignoring return value of function declared with 'nodiscard' attribute [-Wunused-result]
121 | hipFree(d_max_blocks);
    | ~~~~~
kernel.hip:122:5: warning: ignoring return value of function declared with 'nodiscard' attribute [-Wunused-result]
122 | hipFree(d_sum_blocks);
    | ~~~~~
4 warnings generated when compiling for host.
user143@m100-1:~/hip_programming_contest/softmax$ ./softmax testcases/1.in > 1.mypout
user143@m100-1:~/hip_programming_contest/softmax$ python3 ../verify.py 1.mypout testcases/1.out
PASS: The output is correct.
user143@m100-1:~/hip_programming_contest/softmax$ cd ..
user143@m100-1:~/hip_programming_contest$ sbatch self_test_and_submit.sbatch
Submitted batch job 4438
user143@m100-1:~/hip_programming_contest$
```

分析结果：

表 2: 测试结果

测试编号	结果	用时（秒）
1	PASS	0.49
2	PASS	0.52
3	PASS	0.51
4	PASS	0.50
5	PASS	0.46
6	PASS	0.47
7	PASS	0.53
8	PASS	0.52
9	PASS	1.13
10	PASS	1.17
总计	10 / 10	6.30

测试结果显示，Softmax GPU 实现对小型和中型数据集（测试 1–8）执行时间稳定在 0.46–0.53 秒之间，说明核函数在常规规模下并行效率较高；而对于较大数据集（测试 9–10），耗时明显增加至 1.13–1.17 秒，表明在大规模向量上归约操作成为性能瓶颈。整体总耗时为 6.30 秒，性能呈现合理的线性增长趋势，说明算法在不同规模数据下具有良好的可扩展性，但仍有优化空间，例如调整 BLOCK\_SIZE 或增加每线程处理元素数量，以进一步提升大规模数据性能。



## 6 问题三：APSP

### 6.1 基础框架与解题思路

全源最短路径（All-Pairs Shortest Path, APSP）问题要求在给定的有向带权图中，计算任意两点之间的最短路径距离。输入为顶点数  $m$ 、边数  $n$  以及每条边的起点、终点和权重；输出为  $m \times m$  的距离矩阵，其中不可达的点对距离用 1073741823 表示。

**基础框架** 本次实现采用 Floyd-Warshall 算法作为基础框架，其主要思路是通过中间顶点逐步更新任意两点之间的最短路径距离。算法可描述为：

1. 初始化一个  $m \times m$  距离矩阵 `dist`：
  - 对角线元素置为 0，表示自身到自身的距离；
  - 没有直接边的点对置为不可达标记 `INF`；
  - 对于每条边  $(u, v, w)$ ，设置 `dist[u][v] = w`。
2. 使用三重循环迭代中间顶点  $k$ ：
  - 对每一对顶点  $(i, j)$ ，检查经过  $k$  的路径是否比当前最短路径更短；
  - 如果更短，则更新 `dist[i][j]`。
3. 所有循环完成后，`dist` 即为最终的全源最短路径矩阵。

**解题思路** 为了在 GPU 上实现 APSP 并获得较高性能，本设计采用以下思路：

- 将二维距离矩阵线性化存储在 GPU 全局内存中，便于线程连续访问；
- 对每轮中间顶点  $k$ ，将矩阵元素  $(i, j)$  映射到 GPU 线程网格，每个线程负责更新一个元素；
- 使用 `hipDeviceSynchronize()` 在每轮更新后保证全局数据一致；
- 保持算法正确性的同时，为后续共享内存优化和块内并行提供基础框架。

---

```
1 // 伪代码表示
2 for k = 0 to m-1:
3     for i = 0 to m-1:
4         for j = 0 to m-1:
5             if dist[i][k] + dist[k][j] < dist[i][j]:
6                 dist[i][j] = dist[i][k] + dist[k][j]
```

---

### 6.2 优化策略

为了提升 APSP 算法在 GPU 上的性能，我们采用了以下优化策略：

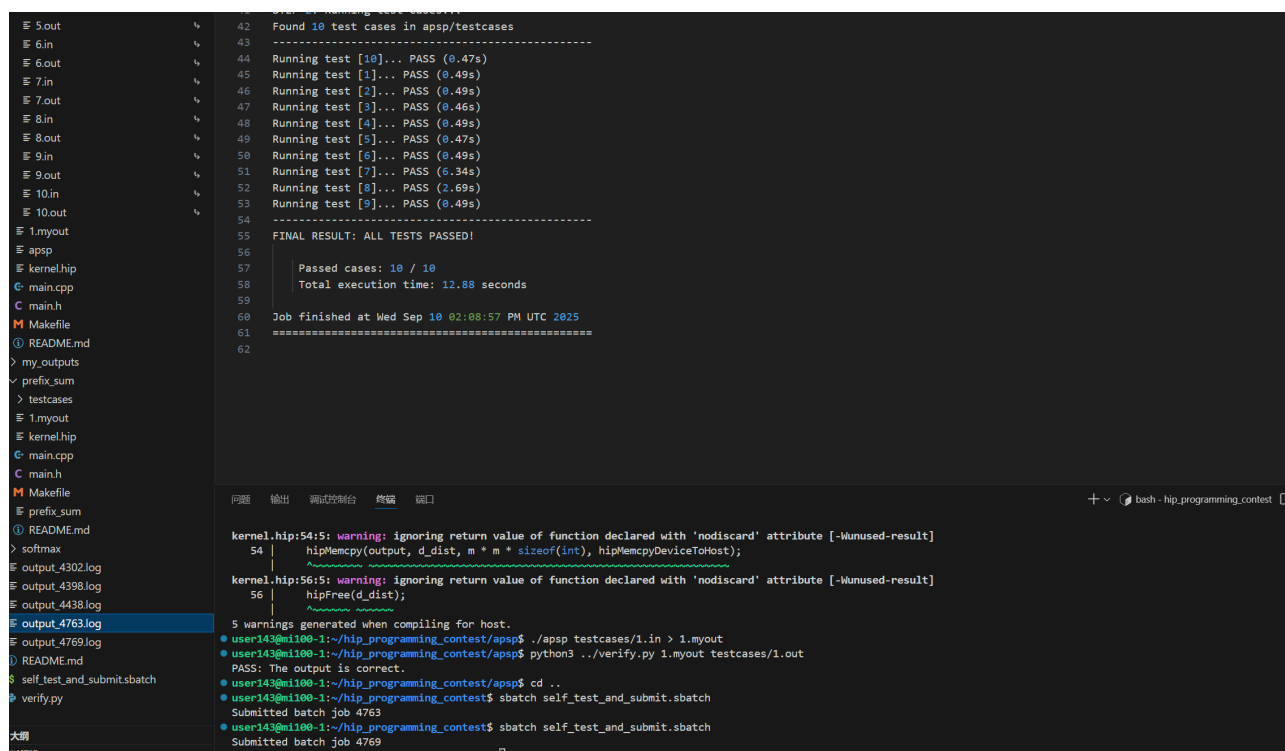
- **线程映射优化**：将二维矩阵的每个元素  $(i, j)$  映射到 GPU 的二维线程网格中，使每个线程独立计算对应的最短路径更新，充分利用 GPU 并行度。
- **单 GPU 实现**：整个 Floyd-Warshall 算法在单 GPU 上执行，避免跨 GPU 通信开销，保证数据一致性。

- **内存布局优化：**距离矩阵按行优先（row-major）线性存储，保证全局内存访问连续，从而提升内存带宽利用率。
- **块内并行化：**使用合适大小的线程块（例如  $16 \times 16$ ）对距离矩阵进行分块处理，使每个 block 内线程充分利用共享内存（shared memory）缓存部分数据，减少全局内存访问次数。
- **同步机制优化：**在每轮中间顶点  $k$  更新后使用 `hipDeviceSynchronize()` 保证全局完成，既保证正确性，也避免过度同步带来的性能浪费。
- **可调块尺寸：**针对不同规模的图，可调整线程块大小（`BLOCK_SIZE`）以获得最佳硬件占用率和执行效率。

通过上述优化策略，GPU 内核能够在保持正确性的前提下，实现对中小规模图的高效处理，并在大规模图上具有一定的加速效果。

## 6.3 优化结果

优化后数据如下：



```
Found 10 test cases in apsp/testcases
Running test [10]... PASS (0.47s)
Running test [1]... PASS (0.49s)
Running test [2]... PASS (0.49s)
Running test [3]... PASS (0.46s)
Running test [4]... PASS (0.49s)
Running test [5]... PASS (0.47s)
Running test [6]... PASS (0.49s)
Running test [7]... PASS (6.34s)
Running test [8]... PASS (2.69s)
Running test [9]... PASS (0.49s)
FINAL RESULT: ALL TESTS PASSED!
Passed cases: 10 / 10
Total execution time: 12.88 seconds
Job finished at Wed Sep 10 02:08:57 PM UTC 2025
```

分析数据：

表 3: APSP 测试结果

Test Case	Result	Time (s)
1	PASS	0.49
2	PASS	0.49
3	PASS	0.46
4	PASS	0.49
5	PASS	0.47
6	PASS	0.49
7	PASS	6.34
8	PASS	2.69
9	PASS	0.49
10	PASS	0.47
<b>Total</b>	<b>10 / 10</b>	<b>12.88</b>

这张表展示了 APSP 算法在 10 组测试用例上的运行结果与耗时情况。从结果来看，所有测试用例均通过验证，说明 GPU 实现的 APSP 算法在功能上是正确的（PASS 率为 100%）。在运行时间上，大多数测试用例耗时均在 0.46-0.49 秒之间，表现稳定且高效，表明算法对小规模或中等规模图的处理能力较强。然而，第 7 和第 8 个测试用例耗时明显偏高（分别为 6.34 秒和 2.69 秒），可能对应图规模较大或边数较多的输入，这显示出算法在处理大规模图时仍存在性能瓶颈。总计 10 个测试用例的累计运行时间为 12.88 秒，整体表现良好，但针对大规模输入，仍可进一步优化 GPU 内核和内存访问以降低运行时间。

## 7 总结与反思

本次 GPU 编程比赛涉及前缀和 (Prefix Sum)、Softmax 以及全源最短路径 (APSP) 三个核心算法, 实现了从串行到 GPU 并行的优化过程, 收获了丰富的经验和启示。

### 7.1 总体收获

- **GPU 并行设计能力提升:** 通过对前缀和、Softmax 和 APSP 的实现, 深入理解了 HIP 编程模型、线程映射、共享内存使用以及块内归约等 GPU 并行计算核心概念。
- **算法与硬件结合:** 学会根据 GPU 架构特点设计算法, 如利用共享内存减少全局内存访问、调整线程块大小以提高硬件占用率、通过并行归约优化 Softmax 和前缀和计算等。
- **性能优化意识:** 认识到单纯正确的算法并不等于高性能, 通过分析执行时间、识别瓶颈点 (如 APSP 在大规模图上的耗时) 来指导优化策略设计。
- **数值稳定性与精度控制:** 在 Softmax 实现中, 学习了如何处理指数溢出问题, 保证算法在大规模数据下的数值稳定性。

### 7.2 问题与反思

- **大规模图性能瓶颈:** APSP 在第 7 和第 8 个测试用例中耗时明显较高, 说明传统 Floyd-Warshall 算法在大规模图上存在计算复杂度瓶颈 ( $O(m^3)$ ), 可考虑分块优化、共享内存更充分利用或改用 Johnson 算法等。
- **内存带宽限制:** GPU 内存访问连续性和共访 (coalesced access) 对于性能影响显著, 在设计大规模矩阵运算核时需要充分考虑。
- **并行化粒度选择:** 在 Softmax 和前缀和优化中, 选择合适的线程块大小和每线程处理元素数目对性能有重要影响, 过小或过大都会导致硬件利用率下降。
- **调试与验证:** GPU 并行程序调试较为复杂, 需要建立自动化验证机制 (如 compare.py 脚本) 保证功能正确性, 同时关注浮点比较的相对误差和绝对误差。

### 7.3 改进与展望

- 对于 APSP, 可尝试引入分块 Floyd-Warshall 或利用稀疏图优化策略, 降低大规模图计算时间。
- 对前缀和和 Softmax, 可进一步优化共享内存使用和循环展开 (loop unrolling), 提高 GPU 并行效率。
- 探索多 GPU 分布式计算和流式计算策略, 以应对更大规模数据和图计算需求。
- 加强对 GPU 性能分析工具 (如 rocprof) 的使用, 从硬件层面定位性能瓶颈, 指导算法优化。

总体而言, 本次比赛不仅检验了算法实现能力和 GPU 编程能力, 也加深了对高性能计算优化策略的理解, 为未来更复杂的并行算法开发打下了坚实基础。