南開大學
Nankai University

计算机学院

并行程序设计期末报告

SYCL 课程学习

姓名：刘家骥

学号：2211437

专业：计算机科学与技术

2025 年 3 月 23 日

目录

1 01_oneAPI_intro

本次课程的学习目标是

- 了解 oneAPI 编程模型如何解决异构计算环境中的编程挑战。
- 学会使用 oneAPI 项目来优化工作流。
- 理解 SYCL 语言和编程模型。
- 熟悉在课程中使用 Jupyter notebooks 进行训练。

课程通过具体示例（如 SYCL Hello World）演示了如何编写和运行 SYCL 程序，并提供了详细的编译和执行步骤。

代码运行结果如图1.1所示

图 1.1: 1_2

通过学习示例代码，我了解了 oneAPI 和 SYCL 的基础知识，学习了如何在异构计算环境中编写高效的并行代码。

2 02_SYCL_Program_Structure

本次课程的学习目标是

- 了解 SYCL 的基本类及其作用。
- 学会如何使用设备选择器将计算任务卸载到不同设备。
- 理解基本并行内核和 ND Range 内核的区别及其使用场景。
- 掌握在 SYCL 程序中使用统一共享内存（USM）或缓冲区访问器内存模型的方法。
- 通过实际操作构建一个简单的 SYCL 应用程序。

课程学习到了包括 SYCL 类、设备选择、队列管理、内核编程、内存模型。

使我较为深入的理解了 SYCL 编程模型，学习了设备选择器、队列管理、并行内核编程以及两种内存模型的应用。

两种内存模型为统一共享内存模型（USM）：主机和设备共享相同的内存空间，数据可以在主机和设备之间无缝访问；缓冲区内存模型：数据通过缓冲区在主机和设备之间传输，访问器用于管理缓冲区的数据访问。

并通过向量加法实验练习巩固了学习成果

图 2.2: 2_1

代码运行，如图2.3所示

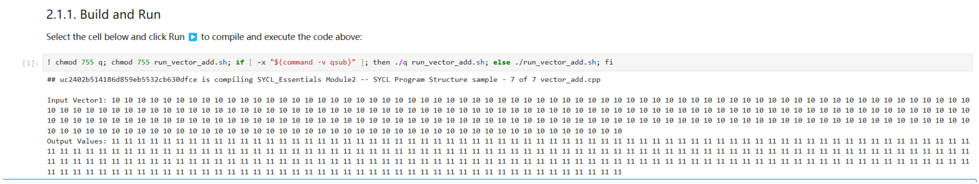

图 2.3: 2_2

3 03_SYCL_Unified_Shared_Memory

本次课程的学习目标是

- 使用 SYCL2020 的新特性如统一共享内存简化编程。
- 理解使用 USM 进行隐式和显式内存移动的方式。
- 在内核任务之间以优化的方式解决数据依赖问题。

统一共享内存（USM）是一种基于指针的内存管理方式，适用于 SYCL。对于习惯使用 malloc 或 new 分配数据的 C 和 C++ 程序员来说，USM 非常熟悉。USM 简化了将现有 C/C++ 代码移植到 SYCL 的过程。

学习了关于 USM 的基础知识，包括 USM 的定义、类型、代码示例（隐式和显式数据移动），以及数据依赖管理。课程还包括了实际操作练习，帮助我们巩固所学内容

隐式数据移动：使用 malloc_shared 或 malloc_host 进行分配，数据可以在主机和设备之间自动迁移。

显式数据移动：使用 malloc_device 分配内存，并显式地在主机和设备之间移动数据。
如图3.4所示

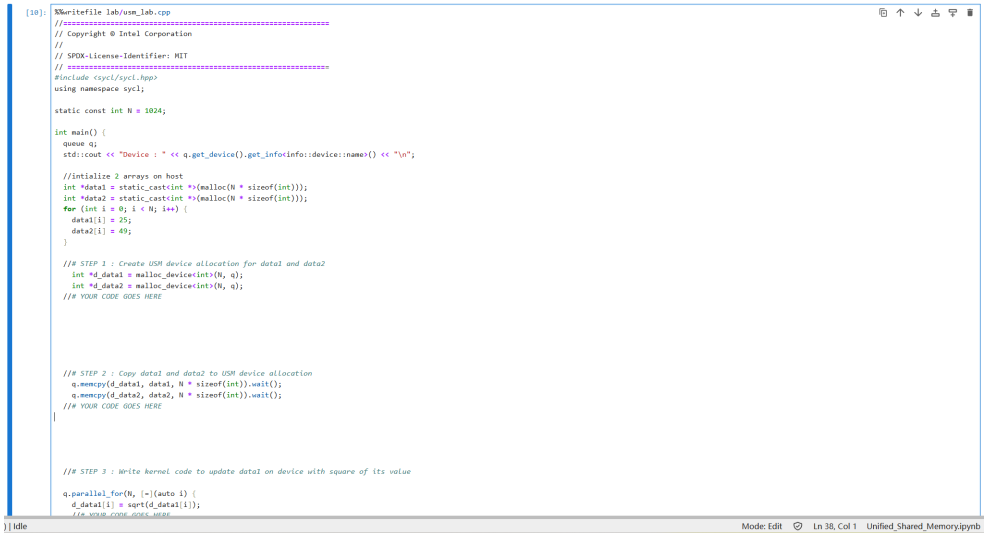

图 3.4: 3_1

通过课程中的实验练习，我实际操作了一个简单的 SYCL 应用程序，练习了如何使用 USM 进行隐式和显式的数据移动，并理解了如何管理内核任务之间的数据依赖。

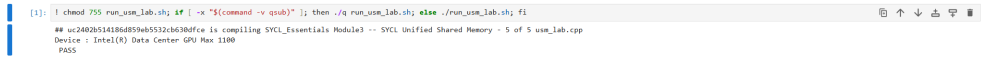

图 3.5: 3_2

通过这门课程的学习，我理解了 SYCL 的统一共享内存模型，掌握了隐式和显式数据移动的使用方法，并尝试了在内核任务之间解决数据依赖问题。通过实际编写代码，我不仅巩固了理论知识，还提升了实际编程能力，为今后在异构计算环境中编写高效并行代码奠定了坚实基础。

4 04_SYCL_Sub_Groups

本次课程的学习目标是

- 了解在 SYCL 中使用子组的优势
- 利用子组算法提高性能和生产力
- 使用子组随机播放操作可避免显式内存操作

本次实验的练习是，使用子组（sub-group）来计算数组的子组和，并将这些和再合并成一个总和。
学习了如何在 SYCL 中使用子组操作，如何进行数据归约，以及如何在内核中和主机端正确地处理数据。这个过程帮助我更深入地理解了并行编程中的数据局部性和并行计算的效率问题。
学习了如何编写内核任务以利用子组（sub-group）进行并行计算。具体来说，我使用了 nd_range 和 nd_item 来定义和管理并行工作项和工作组

```
1 q.parallel_for(nd_range<1>(N, B), [=](nd_item<1> item) {
2   auto sg = item.get_sub_group();
```

```
3     auto i = item.get_global_id(0);
```

使用 nd_range 和 nd_item 可以更好地控制并行执行的范围和粒度

```
static constexpr size_t N = 1024; // global size
static constexpr size_t B = 256;  // work-group size
static constexpr size_t S = 32;   // sub-group size

int main() {
  queue q;
  std::cout << "Device : " << q.get_device().get_info<info::device::name>() << "\n";

  // allocate USM shared allocation for input data array and sg_data array
  int *data = malloc_shared<int>(N, q);
  int *sg_data = malloc_shared<int>(N/S, q);

  // initialize input data array
  for (int i = 0; i < N; i++) data[i] = i;
  for (int i = 0; i < N; i++) std::cout << data[i] << " ";
  std::cout << "\n\n";

  // Kernel task to compute sub-group sum and save to sg_data array

  // STEP 1 : set fixed sub_group size of value S in the kernel below
  q.parallel_for(nd_range<1>(N, B), [=](nd_item<1> item) {
    auto sg = item.get_sub_group();
    auto i = item.get_global_id(0);

    // STEP 2: Add all elements in sub_group using sub_group reduce
    int sg_sum = reduce_over_group(sg, data[i], plus<>());

    // STEP 3 : save each sub-group sum to sg_data array
    if (sg.get_local_id() == 0) {
      sg_data[item.get_group(0)] = sg_sum;
    }
  }).wait();

  // print sg_data array
  for (int i = 0; i < N/S; i++) std::cout << sg_data[i] << " ";
  std::cout << "\n";

  // STEP 4: compute sum of all elements in sg_data array
  int sum = 0;
  for (int i = 0; i < N/S; i++) {
    sum += sg_data[i];
  }

  std::cout << "\nSum = " << sum << "\n";

  // free USM allocations
```

图 4.6: 4_1

运行结果如图4.7所示

1.11.0.1.构建和运行

选择下面的单元格，然后单击"运行"以编译并执行代码:

```
[12]: ! chmod 755 run_sub_group_lab.sh; if [ -x "$(command -v qsub)" ]; then ./q run_sub_group_lab.sh; else ./run_sub_group_lab.sh; fi

## uc2402b514186d859eb5532cb630dfce is compiling SYCL_Essentials Module4 -- SYCL Sub Groups - 7 of 7 sub_group_lab.cpp
Device : Intel(R) Data Center GPU Max 1100
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 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

1520 15856 23024 30192 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Sum = 70592
```

图 4.7: 4_2

通过本次学习和示例代码的撰写，我了解了如何利用子组进行并行计算，接触到了 SYCL 编程的具体方法，增强了我对并行计算和异构计算的理解。

5 学习心得

通过这一系列的学习和实验，我总体上了解了 SYCL 的基本语法和编程模型，并且简单实现了在 SYCL 使用子组来进行并行计算，提升计算性能，使我对并行计算模型有了更深入的理解。