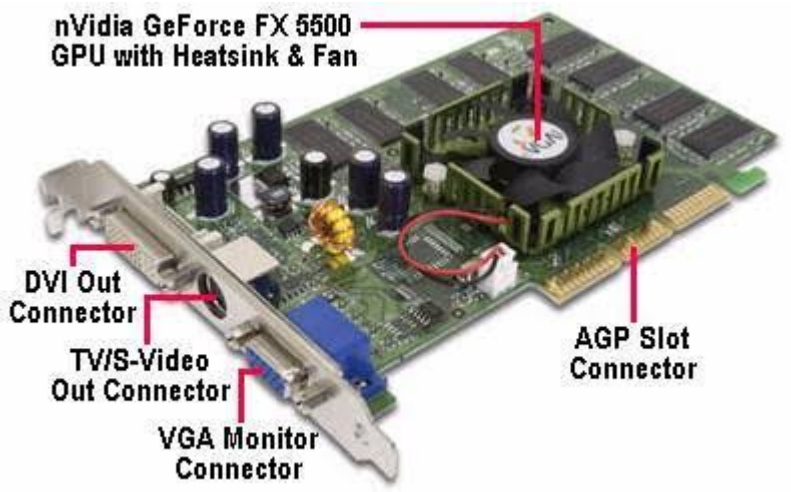
Excercise 5 Introduction to CUDA

สิ่งที่ต้องใช้ 1. WinSCP 2. Terminal 3. Code Editor 4. CUDA (nvcc or Nvidia compiler) 5. **สไลด์อาจารย์ด้วยก็ดีนะ**

GPU **คืออะไร**?

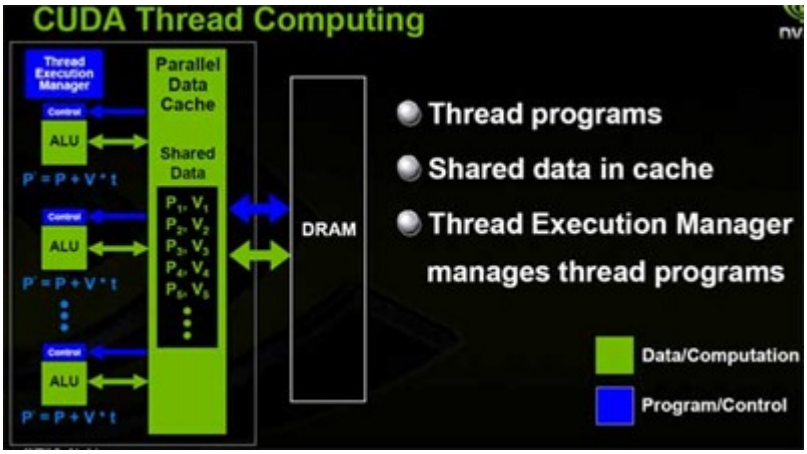
GPU หรือ **Graphic Processing Unit** มันเกิดมาเพื่อประมวลผลกราฟิก 3 มิติโดยเฉพาะมีสองบริษัทตอนนี้คือ **Nvidia** และ **AMD/ATI** ในสมัยก่อน CPU (Central Processing Unit) นั้นจะทำการประมวลผลกราฟิกเป็นหลักทำให้ CPU ทำงานไม่พอต่องานด้านอื่นๆ จึงเกิด GPU ขึ้นมาเพื่อลดภาระของ CPU ได้มากขึ้นและทำงานได้ดียิ่งขึ้นกว่าเดิม



สมัยก่อนจะมีพอร์ตเฉพาะสำหรับกราฟิก ส่วนพอร์ตเชื่อมต่อนั้นจะเรียกว่า **AGP (Accelerated Graphics Port)** ก่อนจะกลายมาเป็น **PCIe (Peripheral Component Interconnection Express)** ซึ้งจะมีตั้งแต่ x16 x8 x4 x1 ช่องเชื่อมต่อสมัยก่อนเรียกว่า **VGA (Video Graphic Array)** ซึ่งช่องต่ออันนี้จะแสดงภาพออกมาในรูปแบบ Matrix Array ซึ่งช่องแต่ละช่องจะเรียกว่า**พิกเซล**ซึ่งบรรจุค่าในอาเรย์เป็นสีทั้งหมด 3 สี แดง น้ำเงิน เหลือง ค่าต่างกันรวมกันจึงเกิดเป็นภาพขึ้นมา

แล้วอะไรคือ CUDA **กันนะ**?

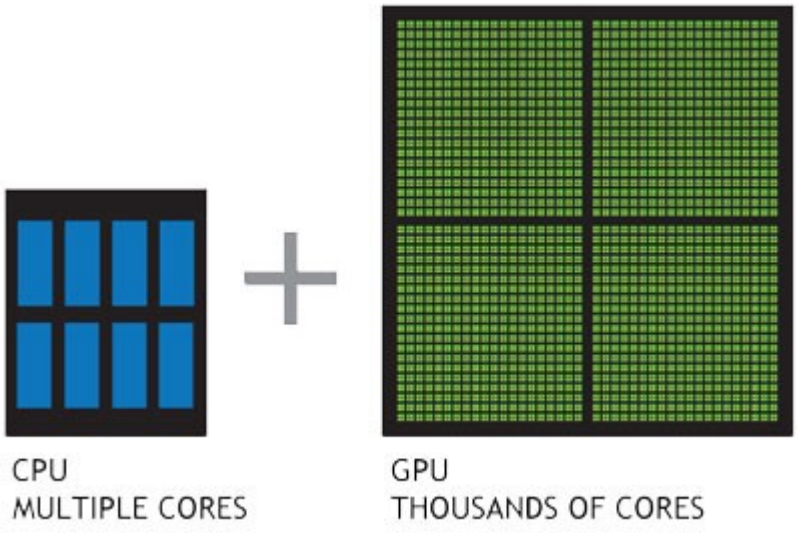
CUDA (Compute Unified Device Architecture) คือ การประมวลผลแบบคู่ขนานและ **Application Programing Interface (API)** นี้เกิดจากบริษัท **Nvidia** เพื่อให้โปรแกรมเมอร์สามารถดึงประสิทธิภาพของการ์ดจอออกมาได้เต็มที่จาก **GPU (Graphic Processing Unit)** หรือ **GPGPU (General-Purpose computing on Graphics Processing Units)**



แล้วทำไมต้องใช้ GPU แทน CPU กันล่ะ

CPU นั้นจะประกอบไปด้วย core เพียงไม่กี่ core จึงเหมาะกับงานในลักษณะที่เรียกว่า Sequential serial processing หรือการประมวลผลแบบลำดับ ในขณะที่ GPU จะประกอบด้วย core ขนาดเล็กจำนวนมากและถูกออกแบบให้มีการกระจายการ

ทำงานในลักษณะ Parallel **หรือการประมวลผลแบบคู่ขนานแบบนี้**



ซึ่ง GPU เหมาะกับงานประเภท **Single Instruction Multiple Data stream (SIMD)**

วิธีการดูเมทริกซ์ใน CUDA

เอาชื่อตัวแปรก่อนเลย

1. threadIdx.x // threadID
2. blockDim.x // block Dimension
3. blockId.x // blockID
4. gridDim.x // grid Dimension

threadIdx.x คือ ตัว Pointer ที่จะชี้/ระบุตำแหน่งภายใน block แกน X ซึ่ง ตรง .x สามามารถเปลี่ยนเป็น Y หรือ Z ได้

blockDim.x จำนวนเธรด (Thread) ในบล็อกแกน X ซึ่ง ตรง .x สามามารถเปลี่ยนเป็น Y หรือ Z ได้

blockId.x คือ Pointer หรือตัวชี้บล็อก (block) ที่ข้างในนั้นมีเธรดบรรจุอยู่ ในแกน X ซึ่ง ตรง .x สามามารถเปลี่ยนเป็น Y หรือ Z ได้

gridDim.x คือ จำนวน threadblocks ในแกน X ซึ่ง ตรง .x สามามารถเปลี่ยนเป็น Y หรือ Z ได้

มาดูตัวอย่าง อาเรย์ 1 มิติ ใน CUDA กัน

```
-------------------------------------------------------
|                                                     |
|     x = 1       x = 2       x = 3       x = 4       |
|  ----------  ----------  ----------  ----------     |
|  |thread 0|  |thread 1|  |thread 2|  |thread 3|     |
|  ----------  ----------  ----------  ----------     |
|                                      ------> แกน X  |
-------------------------------------------------------

threadIdx.x ตัวที่ 1 หรือ thread 0 (X = 1), threadIdx.x = 1 , threadIdx.y =
threadIdx.Z มีค่า 0
threadIdx.x ตัวที่ 3 หรือ thread 2 (X = 3), threadIdx.x = 3 , threadIdx.y =
```

```
threadIdx.Z มีค่า 0
blockDim.x = 4, blockDim.y = 1, blockDim.z = 0 จะมี thread ทั้งหมด 4*1 = 4 เธรด
```

มาดูตัวอย่าง อาเรย์ 2 มิติ ใน CUDA

```
-----------------------------------------
|                                       |
|    ----------  ----------  ----------  |
|    |thread 1|  |thread 2|  |thread 3|  |
|    |x=0, y=0|  |x=1, y=0|  |x=2, y=0|  |
|    ----------  ----------  ----------  |
|    ----------  ----------  ----------  |
|    |thread 4|  |thread 5|  |thread 6|  |
|    |x=0, y=1|  |x=1, y=1|  |x=2, y=1|  |
|    ----------  ----------  ----------  |
|    ----------  ----------  ----------  |
|แ | |thread 7|  |thread 8|  |thread 9|  |
|ก | |x=0, y=2|  |x=1, y=2|  |x=2, y=2|  |
|น | ----------  ----------  ----------  |
|Y V                      ------> แกน X |
-----------------------------------------

threadIdx.x ตัวที่ 1 หรือ thread 1 (threadIdx.x = 1), threadIdx.x = 0 , threadIdx.y =
threadIdx.Z มีค่า 0
threadIdx.x ตัวที่ 6 หรือ thread 6 (threadIdx.x = 2), threadIdx.y = 1 threadIdx.Z มีค่า
0
threadIdx.x ตัวที่ 4 หรือ thread 4 (threadIdx.x = 0), threadIdx.y = 1 threadIdx.Z มีค่า
0
threadIdx.x ตัวที่ 9 หรือ thread 9 (threadIdx.x = 2), threadIdx.y = 2 threadIdx.Z มีค่า
0
blockDim.x = 3, blockDim.y = 3,blockDim.z = 0 ก็คือมี Thread ทั้งหมด 3*3 = 9 thread
```

มาเริ่มที่ข้อ 1. กันเถอะ

1. Use CUDA to assign a value to each element of the array of integers A[256] using 256 threads. Each A[i] is assigned with the value of 2*i, for i = 0 to 255.

โจทย์ข้อนี้ให้เราใช้ CUDA เพื่อกำหนดค่าในอาเรย์ตำแหน่งโดยใช้ อาเรย์ขนาด 256 และใช้เธรดขนาด 256 เธรด โดยอาเรย์ A ที่มีขนาด 256 นั้นในแต่ละตำแหน่งจะถูกกำหนดค่าโดย i * 2 โดยที่ i เริ่มตั้งแต่ 0 จนถึง 255

```
#include <stdio.h>
#define T 256 // As Threads

__global__ void vecMultiply(int *A) {
        int i = threadIdx.x;
        A[i] = A[i] * 2; // ตำแหน่งจะถูกกำหนดค่าโดย i * 2 โดยที่ i เริ่มตั้งแต่ 0 จนถึง 255
}

int main (int argc, char *argv[])
```

```
{
        int i;
        int size = T*sizeof(int);
        int a[T], *devA;
        for (i=0; i< T; i++)
        {
                a[i] = i + 1;
        }
        cudaMalloc( (void**)&devA,size);
        cudaMemcpy( devA, a, size, cudaMemcpyHostToDevice);

        vecMultiply<<<1, T>>>(devA); // ตรงนี้เรียกว่า Kernel launch
        printf("Before\n");
        for (i=0; i< T; i++)
        {
                printf("%d ", a[i]);
        }
        printf("\n");

        cudaMemcpy(a, devA, size, cudaMemcpyDeviceToHost);
        cudaFree(devA);
        printf("After\n");
        for (i=0; i < T; i++)
    {
                printf("%d ",a[i]);
        }
        printf("\n");

}
```

ผลลัพธ์ที่ออกมาจะประมาณนี้

vecMultiply<<<1, T>>>(devA); ตรงนี้ใน **Nvidia CUDA** จะเรียกว่า **kernel Launch** ซึ่งค่า **1** ตรงนี้หมายถึง จำนวนกริดของเธรดบล็อก (Grid number of Thread Blocks) และ ค่า **T** หมายถึง เธรดบล็อกที่มีขนาดเป็น T ตัวในการทำงานแบบคู่ขนาน (thread block has T parallel threads)ถ้าเราจะมองให้เห็นภาพกว่านี้อีกนิ๊ดด

```
mykernel<<<blocks, threads, shared_mem, stream>>>(args);
vecMultiply<<<1, T>>>(devA);
```

คือตรง mykernel **อะมันเป็นคำสั่งหรือ** **kernel launch** **เพื่อส่งข้อมูลไปหา** Device (CUDA) **ประมวลผลลัพธ์มาให้โดยเทียบกับ** **vecMultiply** **มันก็คือ** Kernel launch **เช่นกันโดยค่า** **1** **คือ** **blocks** **และ** **T** **คือ** **Threads Blocks** **นั่นเอง**

ตัวอย่าง

```
__global__ void vecMultiply(int *A) { // ตรงนี้เรียกว่า Kernel
        int i = threadIdx.x;
        A[i] = A[i] * 2; // ตำแหน่งจะถูกกำหนดค่าโดย i * 2 โดยที่ i เริ่มตั้งแต่ 0 จนถึง 255
}
vecMultiply<<<1, T>>>(devA); // ตรงนี้เรียกว่า Kernel launch
```

Function **ที่** CUDA **สามารถติดต่อกับ** Devices **ได้**

```
__host__ // เรียกใช้ได้เฉพาะบน Host และรันได้แค่ Host เท่านั้น
__global__ // เรียกใช้จาก Host ไปรันบน Device (CUDA)
__device__ // เรียกใช้จาก Device ไปรันบน Device เท่านั้น (CUDA)
```

วิธีส่งโค้ดไปคำนวณบน CUDA Device

วิธีมันจะพิสดารกว่าปกติหน่อยเพราะมี **Memory** **ที่แยกกันออกมาเลยต้องกำหนดขนาด** Memory **แล้วค่อยโยนโค้ดไปคิดแล้วโยนผลลัพธ์กลับมาหาเครื่อง วิธีก็ตามนี้**

```
1.  Allocate Memory บน Device # cudaMalloc( (void**)&devA,size);
2.  Transfer ข้อมูลจาก Host ไปยัง Device # cudaMemcpy( devA, a, size,
cudaMemcpyHostToDevice);
3.  Kernel Launch  # vecMultiply<<<1, T>>>(devA);
4.  Transfer ข้อมูลกลับจาก Device ไปยัง Host # cudaMemcpy(a, devA, size,
cudaMemcpyDeviceToHost);
5.  Free Memory บน Device # cudaFree(devA);
```

วิธีคอมไพล์ไฟล์ CUDA (.cu)

ใช้คำสั่งตามนี้

```
nvcc hello.cu -o hello # เพื่อคอมไพล์
nvcc -o hello hello.cu # หรือแบบนี้ก็ย่อมได้
```

วิธีรัน CUDA

วิธีรันคล้ายๆกับการรันภาษาซีบน Unix **เลยแค่**

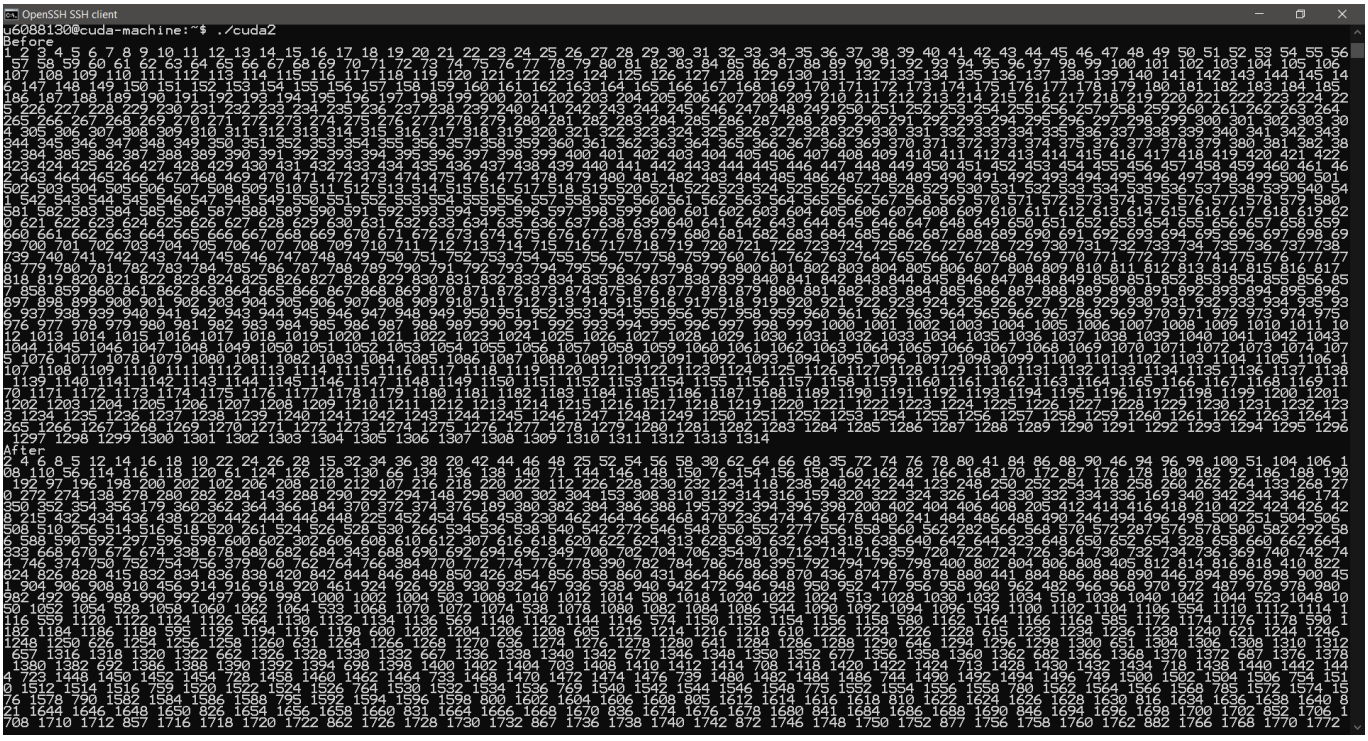
```
./ชื่อไฟล์ที่จะรัน
```

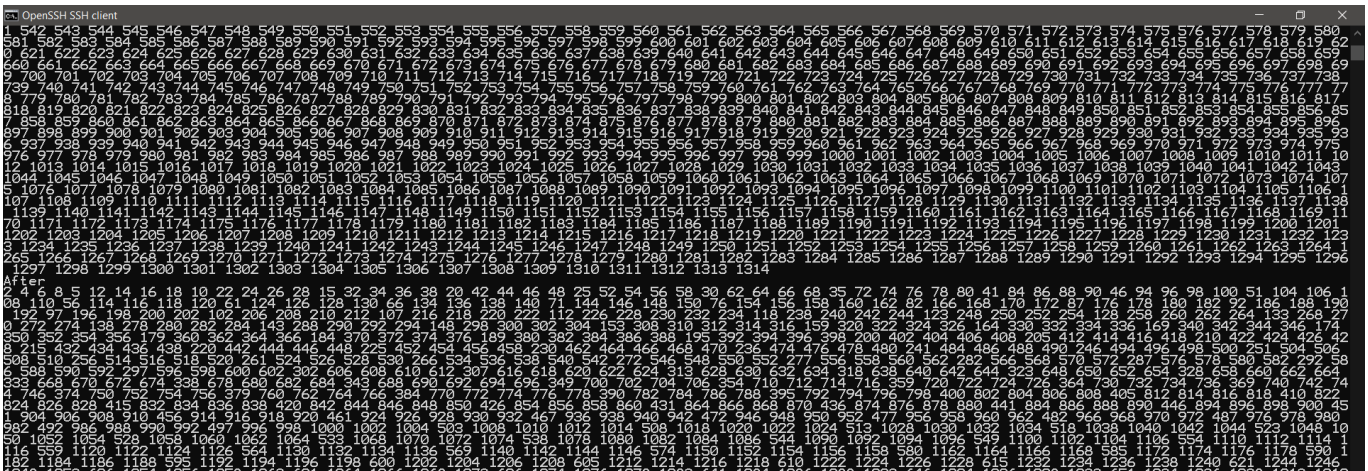
ข้อ 2.

Repeat Question 1 with the array A[1314], using only 256 threads.
ก็คือเอาโค้ดจากข้อ 1 มาแก้เพิ่มโดยกำหนดขนาดอาเรย์ A ให้มีขนาดเป็น 1314 และใช้เธรดแค่ 256 ตัวเท่านั้น

```
#include <stdio.h>

#define T 256 // As Threads
#define ArraySize 1314

__global__ void vecMultiply(int *A)
{
        int i;
        int threadID = threadIdx.x;
        int start = (threadID * ArraySize) / 256;
        int end = ( ( (threadID + 1 ) * ArraySize) / 256) - 1;
        for(i = start ; i < end ; i++)
        {
                A[i] = A[i] * 2;
        }
}
int main (int argc, char *argv[])
{
        int i;
        int size = ArraySize*sizeof(int);
        int a[size], *devA; // ตรงนี้ a[size] จะกำหนดขนาดด้วย 1314 เรียบร้อยแล้ว
        for (i=0; i< ArraySize; i++)
        {
                a[i] = i + 1;
        }
        cudaMalloc( (void**)&devA,size);
        cudaMemcpy( devA, a, size, cudaMemcpyHostToDevice);

        vecMultiply<<<1, 256>>>(devA); // 1 , 256 mean send each data with total
256 thread blocks
        printf("Before\n");
        for (i=0; i< ArraySize; i++)
        {
                printf("%d ", a[i]);
        }
        printf("\n");

        cudaMemcpy(a, devA, size, cudaMemcpyDeviceToHost);
        cudaFree(devA);
        printf("After\n");
        for (i=0; i < ArraySize; i++)
    {
                printf("%d ",a[i]);
        }
        printf("\n");
}
```

ผลลัพธ์ที่ได้





ข้อ 3.

Given two array of integers A and B, each having size 256 elements, write a CUDA program with the following kernel to copy the elements of array A to array B in reverse order using 256 threads. For example, if input array A = {1, 2, 3, ..., 256}, the output array B = {256, ..., 3, 2, 1}.

```
__global__ void reverseArray (int *A, int *B)
{
```

```
    /* Code to reverse array is here */
}
```

โจทย์ข้อนี้เราจะต้องใช้อาเรย์สองตัวแต่ละตัวมีขนาดเก็บข้อมูลได้ 256 ช่อง โดยให้ใช้การเขียนโดย CUDA เพื่อทำการ Copy ค่าจากอาเรย์ A ไปอาเรย์ B โดยค่าที่ Copy นั้นจะต้องเป็นค่าที่กลับกัน จากท้ายสุดมาหน้าสุด (Reversed)

```
#include <stdio.h>
#define T 256 // As Threads

__global__ void reverseArray(int *A, int *B)
{
        int threadID = threadIdx.x;
        int Reverse = (T - 1) - threadID; // ตรงนี้เอาไว้ทำ Reverse
        B[Reverse] = A[threadID]; // ตรงนี้ก็เช่นกัน
}

int main (int argc, char *argv[])
{
        int i;
        int size = T*sizeof(int);
        int a[T],b[T], *devA,*devB;
        for (i=0; i< T; i++)
        {
                a[i] = i + 1;
        }

        cudaMalloc( (void**)&devA,size);
        cudaMalloc( (void**)&devB,size);
        cudaMemcpy( devA, a, size, cudaMemcpyHostToDevice);
        cudaMemcpy( devB, b, size, cudaMemcpyHostToDevice);

        reverseArray<<<1, T>>>(devA,devB); // 1 , T mean send 1 until total 256
thread blocks
        printf("Before\n");
        for (i=0; i< T; i++)
        {
                printf("%d ", a[i]);
        }
        printf("\n");

        cudaMemcpy(a, devA, size, cudaMemcpyDeviceToHost);
        cudaMemcpy(b, devB, size, cudaMemcpyDeviceToHost);
        cudaFree(devA);
        cudaFree(devB);
        printf("After\n");
        for (i=0; i < T; i++)
    {
                printf("%d ",b[i]);
        }
        printf("\n");
```

```
}
```

ผลลัพธ์ที่ได้

สวยงาม

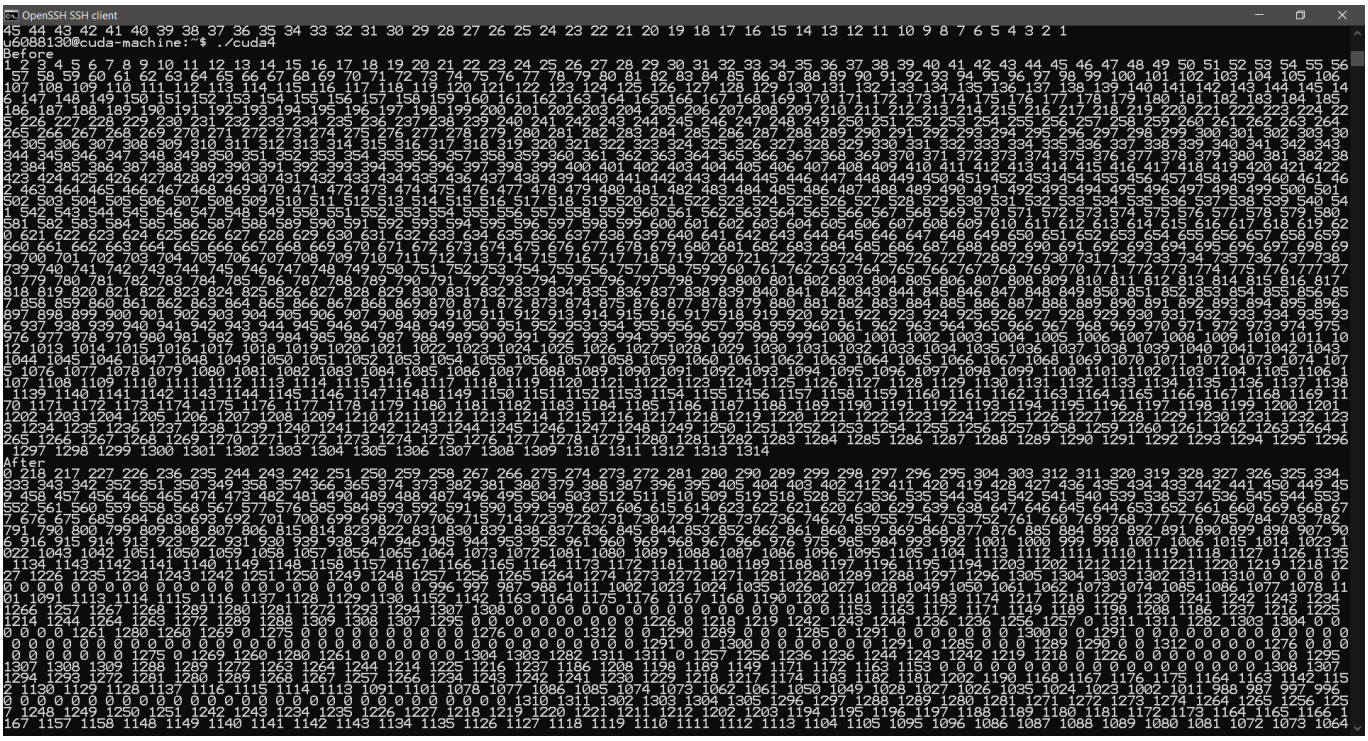
ข้อ 4.

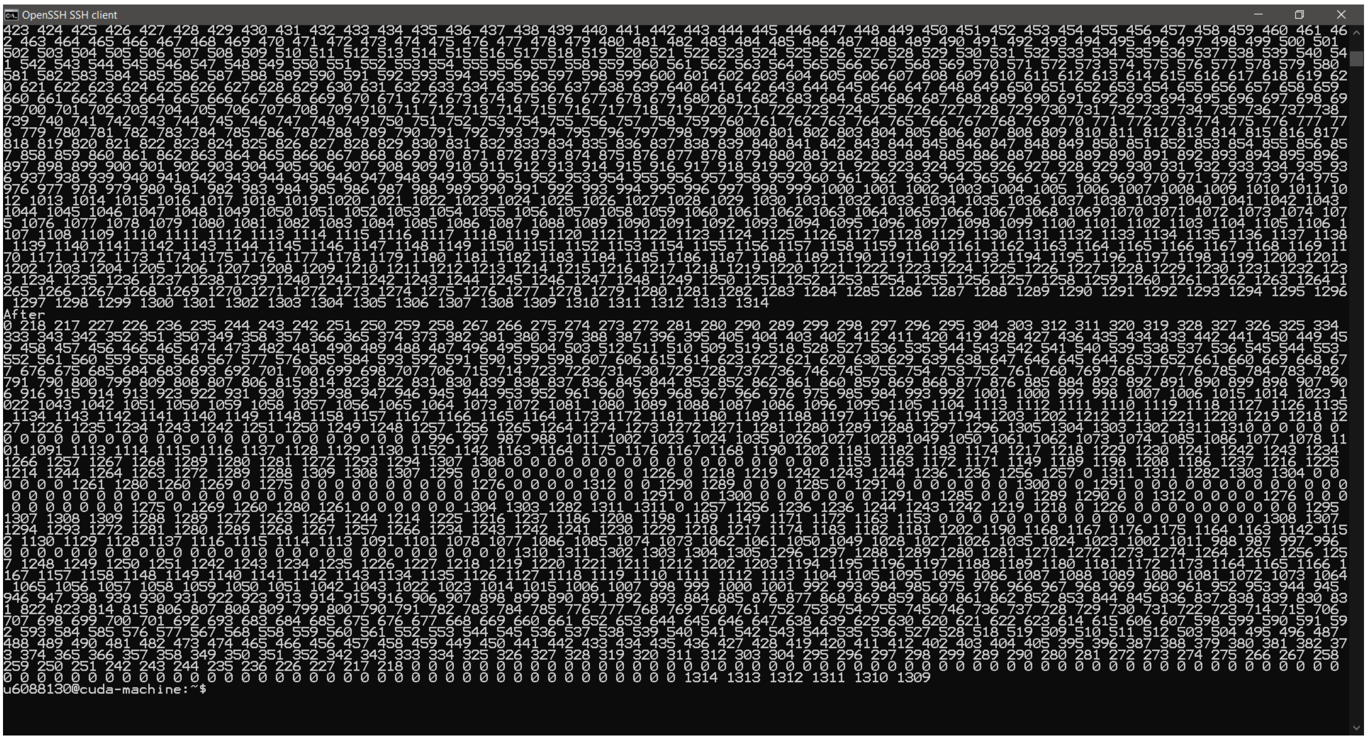
Repeat Question 3 with the array A and B, each having 1314 elements, using only 256 threads. **ก็เหมือนเดิมเพิ่มเติมคือกำหนดขนาดไว้เป็น 1314 และใช้เธรดแค่ 256 ตัวนะจ๊ะ**

```
#include <stdio.h>

#define T 256 // As Threads
#define ArraySize 1314

__global__ void reverseArray(int *A, int *B)
{
        int threadID = threadIdx.x;
        int start = (threadID * ArraySize) / 256;
        int end = ( ( (threadID + 1 ) * ArraySize) / 256) - 1;
        while(end > 0)
        {
                B[end] = A[start];
                end--;
                start++;
        }
}
int main (int argc, char *argv[])
{
        int i;
        int size = ArraySize*sizeof(int);
        int a[ArraySize],b[ArraySize], *devA,*devB;
        for (i=0; i< ArraySize; i++)
        {
                a[i] = i + 1;
        }
```

```
        cudaMalloc( (void**)&devA,size);
        cudaMalloc( (void**)&devB,size);
        cudaMemcpy( devA, a, size, cudaMemcpyHostToDevice);
        cudaMemcpy( devB, b, size, cudaMemcpyHostToDevice);

        reverseArray<<<1, 256>>>(devA,devB); // 1 , 256 mean send each data with
  total 256 thread blocks
        printf("Before\n");
        for (i=0; i< ArraySize; i++)
        {
                printf("%d ", a[i]);
        }
        printf("\n");

        cudaMemcpy(a, devA, size, cudaMemcpyDeviceToHost);
        cudaMemcpy(b, devB, size, cudaMemcpyDeviceToHost);
        cudaFree(devA);
        cudaFree(devB);
        printf("After\n");
        for (i=0; i < ArraySize; i++)
    {
                printf("%d ",b[i]);
        }
        printf("\n");
  }
```

ผลลัพธ์ที่ได้





```
OpenSSH SSH client
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 46
2 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501
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 54
1 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580
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 62
0 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659
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 69
9 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738
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 77
8 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817
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 85
7 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896
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 93
6 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975
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 10
12 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 107
5 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 1
107 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 11
70 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 123
3 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 1
265 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
After
0 218 217 227 226 236 235 244 243 242 251 250 259 258 267 266 275 274 273 272 281 280 290 289 299 298 297 296 295 304 303 312 311 320 319 328 327 326 325 334
333 343 342 352 351 350 349 358 357 366 365 374 373 382 381 380 379 388 387 396 395 405 404 403 402 412 411 420 419 428 427 436 435 434 433 442 441 450 449 45
9 458 457 456 466 465 474 473 482 481 490 489 488 487 496 495 504 503 512 511 510 509 519 518 528 527 536 535 544 543 542 541 540 539 538 537 536 545 544 553
552 561 560 559 558 568 567 577 576 585 584 593 592 591 590 599 598 607 606 615 614 623 622 621 620 630 629 639 638 647 646 645 644 653 652 661 660 669 668 67
7 676 675 685 684 683 693 692 701 700 699 698 707 706 715 714 723 722 731 730 729 728 737 736 746 745 755 754 753 752 761 760 769 768 777 776 785 784 783 782
791 790 800 799 809 808 807 806 815 814 823 822 831 830 839 838 837 836 845 844 853 852 862 861 860 859 869 868 877 876 885 884 893 892 891 890 899 898 907 90
6 916 915 914 913 923 922 931 930 939 938 947 946 945 944 953 952 961 960 969 968 967 966 976 975 985 984 993 992 1001 1000 999 998 1007 1006 1015 1014 1023 1
022 1043 1042 1051 1050 1059 1058 1057 1056 1065 1064 1073 1072 1081 1080 1089 1088 1087 1086 1096 1095 1105 1104 1113 1112 1111 1110 1119 1118 1127 1126 1135
 1134 1143 1142 1141 1140 1149 1148 1158 1157 1167 1166 1165 1164 1173 1172 1181 1180 1189 1188 1197 1196 1195 1194 1203 1202 1212 1211 1221 1220 1219 1218 12
27 1226 1235 1234 1243 1242 1251 1250 1249 1248 1257 1256 1265 1264 1274 1273 1272 1271 1281 1280 1289 1288 1297 1296 1305 1304 1303 1302 1311 1310 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 0 996 997 987 988 1011 1002 1023 1024 1035 1026 1027 1028 1049 1050 1061 1062 1073 1074 1085 1086 1077 1078 11
01 1091 1113 1114 1115 1116 1137 1128 1129 1130 1152 1142 1163 1164 1175 1176 1167 1168 1190 1202 1181 1182 1183 1174 1217 1218 1229 1230 1241 1242 1243 1234
1266 1257 1267 1268 1289 1280 1281 1272 1293 1294 1307 1308 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1153 1163 1172 1171 1149 1189 1198 1208 1186 1237 1216 1225
1214 1244 1264 1263 1272 1289 1288 1309 1308 1307 1295 0 0 0 0 0 0 0 0 0 0 1226 0 1218 1219 1242 1243 1244 1236 1236 1256 1257 0 1311 1311 1282 1303 1304 0 0
0 0 0 0 1261 1280 1260 1269 0 1275 0 0 0 0 0 0 0 0 0 1276 0 0 0 0 1312 0 0 1290 1289 0 0 0 1285 0 1291 0 0 0 0 0 0 1300 0 0 1291 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1291 0 0 1300 0 0 0 0 0 0 0 1291 0 1285 0 0 0 1289 1290 0 0 1312 0 0 0 0 1276 0 0 0
 0 0 0 0 0 0 0 1275 0 1269 1260 1280 1261 0 0 0 0 0 0 1304 1303 1282 1311 1311 0 1257 1256 1236 1236 1244 1243 1242 1219 1218 0 1226 0 0 0 0 0 0 0 0 0 0 1295
1307 1308 1309 1288 1289 1272 1263 1264 1244 1214 1225 1216 1237 1186 1208 1198 1189 1149 1171 1172 1163 1153 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1308 1307
1294 1293 1272 1281 1280 1289 1268 1267 1257 1266 1234 1243 1242 1241 1230 1229 1218 1217 1174 1183 1182 1181 1202 1190 1168 1167 1176 1175 1164 1163 1142 115
2 1130 1129 1128 1137 1116 1115 1114 1113 1091 1101 1078 1077 1086 1085 1074 1073 1062 1061 1050 1049 1028 1027 1026 1035 1024 1023 1002 1011 988 987 997 996
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 1310 1311 1302 1303 1304 1305 1296 1297 1288 1289 1280 1281 1271 1272 1273 1274 1264 1265 1256 125
7 1248 1249 1250 1251 1242 1243 1234 1235 1226 1227 1218 1219 1220 1221 1211 1212 1202 1203 1194 1195 1196 1197 1188 1189 1180 1181 1172 1173 1164 1165 1166 1
167 1157 1158 1148 1149 1140 1141 1142 1143 1134 1135 1126 1127 1118 1119 1110 1111 1112 1113 1104 1105 1095 1096 1086 1087 1088 1089 1080 1081 1072 1073 1064
 1065 1056 1057 1058 1059 1050 1051 1042 1043 1022 1023 1014 1015 1006 1007 998 999 1000 1001 992 993 984 985 975 976 966 967 968 969 960 961 952 953 944 945
946 947 938 939 930 931 922 923 913 914 915 916 906 907 898 899 890 891 892 893 884 885 876 877 868 869 859 860 861 862 852 853 844 845 836 837 838 839 830 83
1 822 823 814 815 806 807 808 809 799 800 790 791 782 783 784 785 776 777 768 769 760 761 752 753 754 755 745 746 736 737 728 729 730 731 722 723 714 715 706
707 698 699 700 701 692 693 683 684 685 675 676 677 668 669 660 661 652 653 644 645 646 647 638 639 629 630 620 621 622 623 614 615 606 607 598 599 590 591 59
2 593 584 585 576 577 567 568 558 559 560 561 552 553 544 545 536 537 538 539 540 541 542 543 544 535 536 527 528 518 519 509 510 511 512 503 504 495 496 487
488 489 490 481 482 473 474 465 466 456 457 458 459 449 450 441 442 433 434 435 436 427 428 419 420 411 412 402 403 404 405 395 396 387 388 379 380 381 382 37
3 374 365 366 357 358 349 350 351 352 342 343 333 334 325 326 327 328 319 320 311 312 303 304 295 296 297 298 299 289 290 280 281 272 273 274 275 266 267 258
259 250 251 242 243 244 235 236 226 227 217 218 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 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 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 0 0 0 0 0 1314 1313 1312 1311 1310 1309
u6088130@cuda-machine:~$
```