

Exercise 7 Advanced Topics in CUDA

1. From the following sequential Rank sort algorithm that allows duplicate numbers, implement a parallel version in CUDA. เราจะต้องอิงอัลกอริทึมตาม Ranksort โดยที่สามารถ sort เลขที่ซ้ำกันได้

```
for (i = 0; i < n; i++) /* for each number */
{
    x = 0; for (j = 0; j < n; j++) /* count number less than it */
        if ((a[i] > a[j]) || (a[i] == a[j]) && (j < i))
            x++;
    b[x] = a[i]; /* copy number into correct place */
}
```

เราจะ implement ตัว rank sort นี้เข้ามาใช้ในโค้ดของเรา

```
#include <stdio.h>
#include <stdlib.h>
#include <thrust/sort.h>
#define thread 512
__global__ void rank_sort(int *data, int *result)
{
    int i,j,position;
    position = 0;
    i = threadIdx.x;
    int self = data[i];

    for(j = 0; j < thread; j++)
    {
        if(( self > data[j]) || (self == data[j]) && (j < i))
        {
            position+=1;
        }
    }
    result[position] = self;
}

int main(int argc, char *argv[]){
    int *arr,*data;
    int i;
    int Data[thread],sort[thread];
    int size = sizeof(int)*thread;
    srand(123);
    printf(" Generate Ok\n");
    for(i = 0; i < thread; i++)
    {
        Data[i] = rand() % 100;
        printf("%d ",Data[i]);
    }
}
```

ผลลัพธ์ที่ได้

[illegible]

- Create an array of random integers in CPU
- Receive an integer to search from a user
- Use CUDA to count the number of occurrences of the integer in the array (Use atomic operation. Note that this exercise ignores synchronization overhead)
- Display the number of occurrences of the input integer found in the array

- 2 / 12

3. ใช้ CUDA ในการนับเลขซ้ำในอาเรย์โดยใช้ Atomic operation (การลดรูปฟังก์ชัน หรือการใช้ลูป โดยใช้เมธอด)
4. แสดงผลลัพธ์ตัวเลขที่มีค่าซ้ำออกมาจากอาเรย์

```
#include <stdio.h>
#include <stdlib.h>
#define num_thread 64
#define thread 16
__global__ void count(int *data,int input, int *result)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if(data[i] == input)
    {
        int a = 1;
        atomicAdd(result,a); // atomicAdd หรือ การเพิ่มค่า โดยเพิ่มค่าจาก a ไปที่ result
        // result += a;
    }
}

int main(int argc, char *argv[]){
    int Data[num_thread], *arr,input,*result;
    int i;
    int resultarr[1];
    int size = sizeof(int)*num_thread;
    srand(123);
    cudaSetDevice(0);
    for(i = 0; i < num_thread; i++)
    {
        Data[i] = rand() % 50;
        printf("%d ",Data[i]);
    }
    printf("Input value to find: ");
    scanf("%d",&input);
    printf("\n");
    cudaMalloc( (void**) &arr, size);
    cudaMalloc( (void**) &result, sizeof(int));

    cudaMemcpy(arr,Data,size, cudaMemcpyHostToDevice);

    count<<<num_thread/thread,thread>>>(arr,input,result);

    cudaMemcpy(resultarr,result,sizeof(int),cudaMemcpyDeviceToHost);

    cudaFree(result);
    cudaFree(arr);

    printf(" Value %d to search occurrences Data found: %d",input,resultarr[0]);

    printf("\n");

    return 0;
}
```

มาตรฐาน Kernel

```
__global__ void count(int *data,int input, int *result)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if(data[i] == input)
    {
        int a = 1;
        atomicAdd(result,a); // atomicAdd หรือ การเพิ่มค่า โดยเพิ่มค่าจาก a ไปที่ result
        // result += a;
    }
}
```

ตรง **Kernel** นี้จะสังเกตว่าเราใช้ **atomicAdd** (Atomic Operation) ในการหาผลลัพธ์ทั้งหมดจากการบวกค่าทั้งหมด

แล้วอะไรคือ Atomic Operation กันล่ะ?

Atomic Operation คือการทำงานด้วยเธรดเดียว (Single Thread) และใช้หน่วยความจำเพียงชนิดเดียวเท่านั้น

Atomic Operation มีกี่แบบ? ** มี 5 แบบ **

1. Addition/subtraction: atomicAdd, atomicSub
2. Minimum/maximum: atomicMin, atomicMax
3. Conditional increment/decrement: atomicInc, atomicDec
4. Exchange/compare-and-swap: atomicExch, atomicCAS
5. More types in Fermi: atomicAnd, atomicOr, atomicXor

Atomic Operation นั้นเป็นการทำงานแบบที่ต้องใช้แค่เธรดเดียวเท่านั้น เธรดอื่นไม่สามารถเข้ามาแก้ไขหรือดูค่าได้ ซึ่ง Atomic Operation นั้นสามารถกัน Race Condition หรือที่เรียกว่าการดึงค่าไปใช้ซ้ำกันหรือเรียกการเรียกพารามิเตอร์ซ้ำซ้อน**

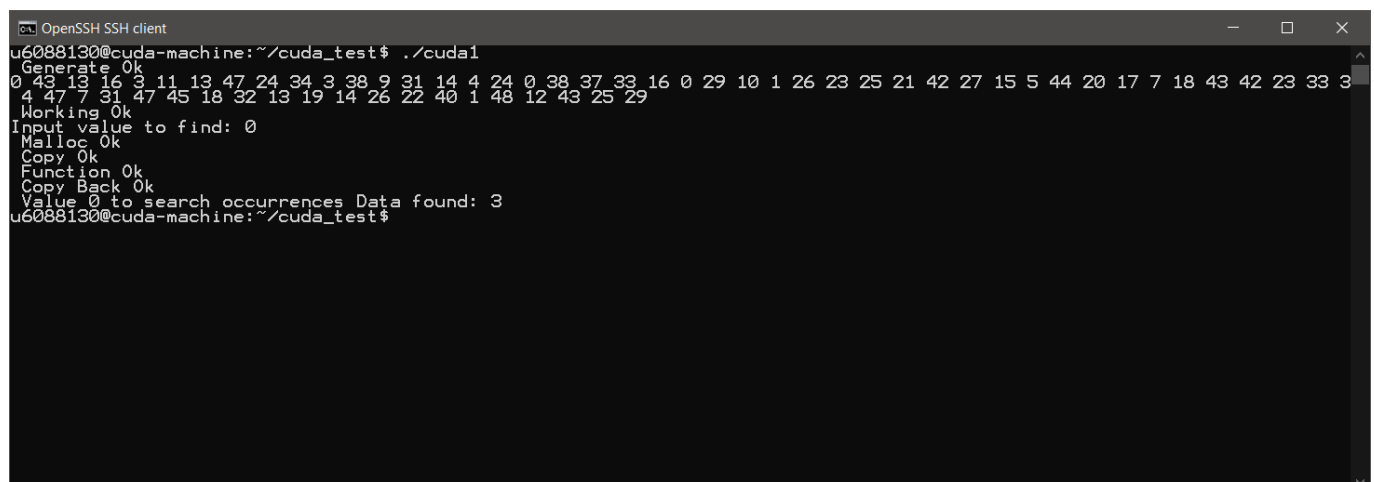
Atomic Operation สามารถ อ่าน/แก้ไข และ เขียนค่าลงไปในหน่วยความจำได้เลยโดยไม่มีการรบกวนจากสิ่งต่างๆที่เกิดขึ้น เช่น การเรียกใช้ค่านั้นๆ

Atomic Operation นั้นในหน่วยความจำแบบใช้ร่วมกัน(Shared memory)

ส่วนการแบ่งหน่วยความจำให้กับทุกฟังก์ชัน(Global memory) นั้นจะใช้ป้องกันการใช้พารามิเตอร์ระหว่างสองเธรดที่ต่างกัน

หลังจากที่ **Kernel call** เรียบร้อยแล้ว ผลลัพธ์จะเท่ากับค่า **input**

ผลลัพธ์ที่ได้



```
u6088130@cuda-machine: ~/cuda_test$ ./cuda1
Generate Ok
0 43 13 16 3 11 13 47 24 34 3 38 9 31 14 4 24 0 38 37 33 16 0 29 10 1 26 23 25 21 42 27 15 5 44 20 17 7 18 43 42 23 33 3
4 47 7 31 47 45 18 32 13 19 14 26 22 40 1 48 12 43 25 29
Working Ok
Input value to find: 0
Malloc Ok
Copy Ok
Function Ok
Copy Back Ok
Value 0 to search occurrences Data found: 3
u6088130@cuda-machine: ~/cuda_test$
```

ข้อ 3. Given a matrix A[16][16], write a CUDA program to find the summation of each row and each column using atomic operation.

เราจะต้องสร้างเมทริกซ์ที่มีขนาด **16 * 16 (A[16][16])** โดยใช้ **CUDA** หาผลลัพท์ของแต่ละแถวและแต่ละหลักโดยใช้ **Atomic Operation**

```
#include <stdio.h>

#define n 16

__global__ void countNumberInArray(int *originalData, int *arrayCount)
{
    int index = threadIdx.x, i;
    int sum = 0;
    if(threadIdx.x < n)
    {
        for(i = 0; i < n; i++)
        {
            sum += originalData[(index * n) + i];
            printf("%3d ", threadIdx.x);
        }
    }
    else
    {
        for(i = 0; i < n; i++)
        {
            sum += originalData[(i * n) + index];
            printf("%3d ", threadIdx.x);
        }
    }
    atomicAdd(&arrayCount[index], sum);
}

int main(int argc, char *argv[])
{
    int totalCount = 2 * n;
    int originalData[n][n], count[totalCount];
    int i = 0;
    int j = 0;

    int *deviceOriginalData, *deviceArrayCount;

    int arrayByteSize = (n * n) * sizeof(int);
    int countArrayByteSize = totalCount * sizeof(int);

    printf("ORIGINAL: \n");
    for(i = 0; i < n; i++)
    {
        for(j = 0; j < n; j++)
        {
            originalData[i][j] = i;
            printf("%3d ", originalData[i][j]);
        }
    }
}
```

```

    }
    printf("\n");
}
printf("\n\n");

cudaMalloc((void**) &deviceOriginalData, arrayByteSize);
cudaMalloc((void**) &deviceArrayCount, countArrayByteSize);
cudaMemcpy(deviceOriginalData, originalData, arrayByteSize,
cudaMemcpyHostToDevice);

dim3 blockDim(totalCount);
countNumberInArray<<<1, blockDim>>>(deviceOriginalData, deviceArrayCount);

cudaMemcpy(count, deviceArrayCount, countArrayByteSize,
cudaMemcpyDeviceToHost);
cudaFree(deviceOriginalData);
cudaFree(deviceArrayCount);

int rowCounts[n], colCounts[n], rowArrayIterator = 0, colArrayIterator = 0;
int rowsum = 0;
int colsum = 0;
int l = 0;
for(l = 0; l < totalCount; l++)
{
    if(l < n)
    {
        rowCounts[rowArrayIterator++] = count[l];
        rowsum += count[l];
    }
    else
    {
        colCounts[colArrayIterator++] = count[l];
        colsum += count[l];
    }
}
printf("TOTAL COUNT ROW\n");
for(l = 0; l < n; l++)
{
    printf("(%d,%3d)", l, rowCounts[l]);
}
printf("\nSum Row: %d\n", rowsum);
printf("\n\nTOTAL COUNT COL\n");
for(l = 0; l < n; l++)
{
    printf("(%d,%3d)", l, colCounts[l]);
}
printf("\nSum Col: %d\n", colsum);
printf("\n");
return 0;
}

```

```
__global__ void countNumberInArray(int *originalData, int *arrayCount)
{
    int index = blockIdx.x * blockDim.x + threadIdx.x, i;

    int sum = 0;
    if(threadIdx.x < n)
    {
        for(i = 0; i < n; i++)
        {
            if(i < n)
            {
                sum += originalData[(index * n) + i];
                // atomicAdd(&arrayCount[index],sum);
            }
            else
            {
                sum += originalData[(index * n) + index];
                // atomicAdd(&arrayCount[index],sum);
            }
            printf("%3d ",threadIdx.x);
        }
    }
    else
    {
        for(i = 0; i < n; i++)
        {
            if(i < n)
            {
                sum += originalData[(index * n) + i];
                // atomicAdd(&arrayCount[index],sum);
            }
            else
            {
                sum += originalData[(index * n) + index];
                // atomicAdd(&arrayCount[index],sum);
            }
            printf("%3d ",threadIdx.x);
        }
    }
    atomicAdd(&arrayCount[index],sum);
}
```

การที่เราใช้ **if-else condition** นั้นเพื่อทำการแบ่งให้เธรด(thread) ทำงานคนละครั้งซึ่งจะแบ่งแบบนี้

[illegible]

โดยตัวเลข 0 - 15 (รวมทั้งหมด 16 threads) นั้นคือการทำส่วน แถว (Row) ก่อน ส่วน 16 - 31 (รวมทั้งหมด 16 threads) จะ
ทำส่วนหลัก (Column) ซึ่งค่าแต่ละหลักรวมกันจะเป็นแบบนี้

```
Row[1] = 0 บวกกัน 16 ครั้ง จะได้ 0
Row[2] = 1 บวกกัน 16 ครั้ง จะได้ 16
.
.
.
Row[16] = 15 บวกกัน 16 ครั้ง จะได้ 240
-----
Column[1] = บวกตั้งแต่ 0 ไปจนถึง 15 จะได้ 120 # จะได้ 120 ทุกแถว (0+1+2+...+16)
column[2] = บวกตั้งแต่ 0 ไปจนถึง 15 จะได้ 120 # จะได้ 120 ทุกแถว (0+1+2+...+16)
.
.
.
column[16] = บวกตั้งแต่ 0 ไปจนถึง 15 จะได้ 120 # จะได้ 120 ทุกแถว (0+1+2+...+16)
```

ผลลัพธ์ที่ได้

[illegible]

ข้อ 4. Write a CUDA program to find min, max and average values from an array of student scores using reduction operation. Assume that the number of students is a power of 2.

ข้อนี้เราก็ต้องใช้ CUDA ในการหาค่าที่น้อยสุด ค่าที่มากที่สุด และค่าเฉลี่ย จากอาเรย์คะแนนของนักเรียนโดยใช้ **reduction operation** (การมัดรวม/หรือยุบข้อมูลโดยใช้ฟังก์ชันเดียวเท่านั้น) โดยนักเรียนมีค่ายกกำลัง 2

```
#include <stdio.h>
#include <stdlib.h>

#define T 256
#define n 1024

__global__ void reduceToSummation(int *originalData, int stride)
{
    int threadId = (blockIdx.x * blockDim.x) + threadIdx.x;
    int idx = 2 * stride * threadId;
    if(idx < n)
    {
        originalData[idx] = originalData[idx] + originalData[idx + stride];
    }
}

__global__ void reduceToMinimum(int *originalData, int stride)
{
    int threadId = (blockIdx.x * blockDim.x) + threadIdx.x;
    int idx = 2 * stride * threadId;
    if(idx < n)
    {
        int min = originalData[idx];
        if(originalData[idx + stride] < min)
        {
            min = originalData[idx + stride];
        }
        originalData[idx] = min;
    }
}

__global__ void reduceToMaximum(int *originalData, int stride)
{
    int threadId = (blockIdx.x * blockDim.x) + threadIdx.x;
    int idx = 2 * stride * threadId;
    if(idx < n)
    {
        int max = originalData[idx];
        if(originalData[idx + stride] > max)
        {
            max = originalData[idx + stride];
        }
        originalData[idx] = max;
    }
}

int main(int argc, char *argv[])
{
    int originalData[n];
```

```

int sum, min, max;
int i;
int *deviceOriginalData;
int arrayByteSize = n * sizeof(int);
printf("ORIGINAL: \n");
for(i = 0; i < n; i++)
{
    originalData[i] = i;
    printf("%3d ", originalData[i]);
}
printf("\n\n");
// Allocates Once for all kernels
cudaMalloc((void**) &deviceOriginalData, arrayByteSize);

// KERNEL 1: Find Average by Finding Summation
cudaMemcpy(deviceOriginalData, originalData, arrayByteSize,
cudaMemcpyHostToDevice);
for(int s = 1; s < n; s *= 2)
{
    reduceToSummation<<<(n + T - 1) / T, T>>>(deviceOriginalData, s);
}
cudaMemcpy(&sum, deviceOriginalData, sizeof(int), cudaMemcpyDeviceToHost);
double realAverage = sum / (double) n;

// KERNEL 2: Find Minimum
cudaMemcpy(deviceOriginalData, originalData, arrayByteSize,
cudaMemcpyHostToDevice);
for(int s = 1; s < n; s *= 2)
{
    reduceToMinimum<<<(n + T - 1) / T, T>>>(deviceOriginalData, s);
}
cudaMemcpy(&min, deviceOriginalData, sizeof(int), cudaMemcpyDeviceToHost);

// KERNEL 3: Find Maximum
cudaMemcpy(deviceOriginalData, originalData, arrayByteSize,
cudaMemcpyHostToDevice);
for(int s = 1; s < n; s *= 2)
{
    reduceToMaximum<<<(n + T - 1) / T, T>>>(deviceOriginalData, s);
}
cudaMemcpy(&max, deviceOriginalData, sizeof(int), cudaMemcpyDeviceToHost);

// Free the memory
cudaFree(deviceOriginalData);

// Print the results
printf("\nAverage is %.2f", realAverage);
printf("\nThe Minimum Number is %d\n", min);
printf("The Maximum Number is %d\n", max);
return 0;
}

```

```
__global__ void reduceToSummation(int *originalData, int stride)
{
    int threadId = (blockIdx.x * blockDim.x) + threadIdx.x;
    int idx = 2 * stride * threadId;
    if(idx < n)
    {
        originalData[idx] = originalData[idx] + originalData[idx + stride];
    }
}
```

kernel ตัวแรกเราจะให้ **Host (CPU)** นั้นสร้างค่าขึ้นมาแล้วส่งต่อให้ **Device (GPU)** ซึ่งจะเรียก **Kernel reduceToSummation** ไปหาค่ารวมทั้งหมดก่อนที่จะส่งกลับไปที่ Host เพื่อหาค่าเฉลี่ย

Kernel ตัวที่สองก็เหมือนตัวแรกแต่จะทำการหาค่าน้อยที่สุดในอาเรย์ออกมาให้

```
__global__ void reduceToMinimum(int *originalData, int stride)
{
    int threadId = (blockIdx.x * blockDim.x) + threadIdx.x;
    int idx = 2 * stride * threadId;
    if(idx < n)
    {
        int min = originalData[idx];
        if(originalData[idx + stride] < min)
        {
            min = originalData[idx + stride];
        }
        originalData[idx] = min;
    }
}
```

Kernel ตัวที่ 3 เหมือนกับตัวที่ 1 และ 2 แต่ตัวที่ 3 จะเป็นการหาค่าที่มากที่สุดในการหา

```
__global__ void reduceToMaximum(int *originalData, int stride)
{
    int threadId = (blockIdx.x * blockDim.x) + threadIdx.x;
    int idx = 2 * stride * threadId;
    if(idx < n)
    {
        int max = originalData[idx];
        if(originalData[idx + stride] > max)
        {
            max = originalData[idx + stride];
        }
        originalData[idx] = max;
    }
}
```

ซึ่ง Kernel ทั้ง 3 ตัวนี้คือ **Reduction Operation** หรือที่เรียกว่าการมักรวมหรือยุบข้อมูลให้อามาคิดในฟังก์ชันเดียว

ผลลัพธ์ที่ได้จะเป็นแบบนี้

```
OpenSSH SSH client
u6088130@cuda-machine:~/cuda_test$ ./cuda3
ORIGINAL:
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

Average is 511.50
The Minimum Number is 0
The Maximum Number is 1023
u6088130@cuda-machine:~/cuda_test$
```