Constant Memory

Apart from all the other memories shown in the previous diagram, we have Constant Memory:

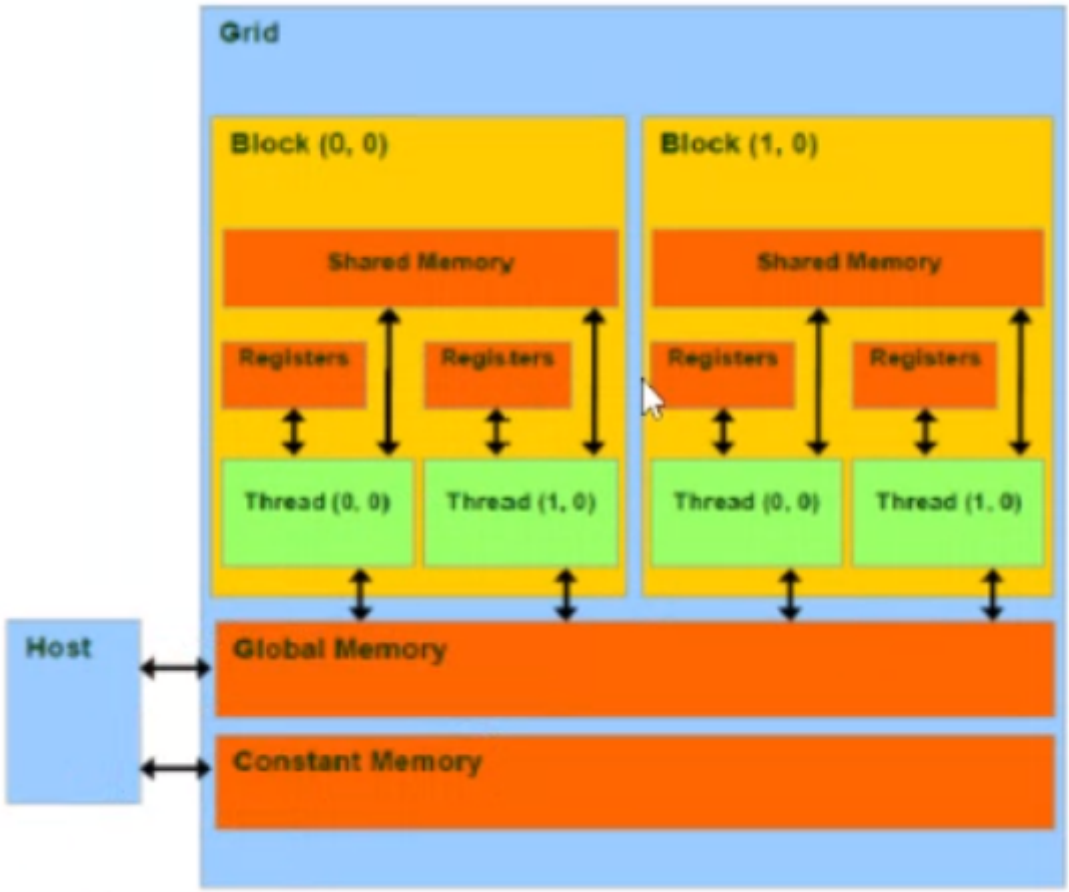

Figure 1: img

The difference with this memory is that it is a **read-only** memory access, thus it is used from data that we just need to read. This memory is inside the Device, but its reservation it's done outside a kernel:

```
1 #define N 32
2 __constant__ int dev_A[N*N];
3 __global__ void kernel(int* dev_A, int* dev_B){
4
5 }
6 int main(){
7
8 }
```

Before using this constant memory, we used to send the information stored in global

memory (*devPtr) as a parameter dev_A, and another vector for the results dev_B. Now, we can replace the reservation of dev_A in the Global Memory. After tne reservation, we used `cudaMemcpy()` to copy the data to dev_A. We also can forget about this step, and instead we transfer the data from the host to dev_A using `cudaMemcpyToSymbol(dev_A, host_A, sizeof(int)*N*N)`: this new function does not require the direction of the transference.

Constant Memory is advised to be used when we only have read-only data, because the memory access is faster than Global Memory.

Implementation

Given a square matrix of size N, output the transpose of such matrix using Constant Memory.

Figure 2: img

Generate a vector in the host: `[1,2,3,4,5,6,7,8,9]`, and copy them to the constant memory using `cudaMemcpyToSymbol(dev_A, host_A, sizeof(int)*N*N)`, now in dev_A. The result will be stored in dev_B as `[1,4,7,2,5,8,3,6,9]`. We do not need to send dev_A as parameter to the kernel, only parameter dev_B is sent. Thus, the result is in Global Memory, in dev_B. Therefore, we only need `cudaMalloc` for dev_B.

- 1 1D grid
- 1 2D block

```
dim3 grid(1);
dim3 block(N, N);
```

- Use validation for the kernel

Solution

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

```

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

#define N 32
__constant__ int dev_A[N * N];

using namespace std;

__host__ void checkCUDAError(const char* msg) {
    cudaError_t error;
    cudaDeviceSynchronize();
    error = cudaGetLastError();
    if (error != cudaSuccess) {
        printf("ERROR %d: %s (%s)\n", error,
           cudaGetErrorString(error), msg);
    }
}

__host__ void validate(int* result_CPU, int* result_GPU) {
    for (int i = 0; i < N * N; i++) {
        if (*result_CPU != *result_GPU) {
            printf("[FAILED] The results are not equal.\n");
            return;
        }
    }
    printf("[SUCCESS] Kernel validation.\n");
    return;
}

__host__ void CPU_transpose(int* vector, int* res) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            res[(i * N) + j] = vector[(N * j) + i];
        }
    }
}

__global__ void GPU_transpose(int* res) {
    int gId = threadIdx.x + (blockDim.x * threadIdx.y);
    res[gId] = dev_A[N * threadIdx.x + threadIdx.y];
}

__host__ void printMtx(int* mtx) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            cout << mtx[(i * N) + j] << " ";
        }
        cout << endl;
    }
}
```

```
54
55 int main() {
56
57     int* dev_B;
58     int* host_B = (int*)malloc(sizeof(int) * N * N);
59     int* cpu_B = (int*)malloc(sizeof(int) * N * N);
60     int* host_A = (int*)malloc(sizeof(int) * N * N);
61
62     cudaMalloc((void**)&dev_B, sizeof(int) * N * N);
63     checkCUDAError("Error at cudaMalloc: dev_B");
64
65     for (int i = 0; i < N * N; i++) {
66         host_A[i] = i + 1;
67     }
68
69     cudaMemcpyToSymbol(dev_A, host_A, sizeof(int) * N * N);
70     checkCUDAError("Error at MemcpyToSymbol");
71
72     dim3 grid(1);
73     dim3 block(N, N);
74     GPU_transpose << < grid, block >> > (dev_B);
75     checkCUDAError("Error at kernel");
76     cudaMemcpy(host_B, dev_B, sizeof(int) * N * N,
           cudaMemcpyDeviceToHost);
77     checkCUDAError("Error at Memcpy host_B <- dev_B");
78
79     CPU_transpose(host_A, cpu_B);
80
81     printf("Input: \n");
82     printMtx(host_A);
83     printf("CPU: \n");
84     printMtx(cpu_B);
85     printf("GPU: \n");
86     printMtx(host_B);
87
88     validate(cpu_B, host_B);
89
90     free(host_B);
91     free(cpu_B);
92     free(host_A);
93     cudaFree(dev_B);
94
95     return 0;
96 }
```

Output

```
GPU:
1 33 65 97 129 161 193 225 257 289 321 353 385 417 449 481 513 545 577 609 641 673 705 737 769 801 833 865 897 929 961 993
2 34 66 98 130 162 194 226 258 290 322 354 386 418 450 482 514 546 578 610 642 674 706 738 770 802 834 866 898 930 962 994
3 35 67 99 131 163 195 227 259 291 323 355 387 419 451 483 515 547 579 611 643 675 707 739 771 803 835 867 899 931 963 995
4 36 68 100 132 164 196 228 260 292 324 356 388 420 452 484 516 548 580 612 644 676 708 740 772 804 836 868 900 932 964 996
5 37 69 101 133 165 197 229 261 293 325 357 389 421 453 485 517 549 581 613 645 677 709 741 773 805 837 869 901 933 965 997
6 38 70 102 134 166 198 230 262 294 326 358 390 422 454 486 518 550 582 614 646 678 710 742 774 806 838 870 902 934 966 998
7 39 71 103 135 167 199 231 263 295 327 359 391 423 455 487 519 551 583 615 647 679 711 743 775 807 839 871 903 935 967 999
8 40 72 104 136 168 200 232 264 296 328 360 392 424 456 488 520 552 584 616 648 680 712 744 776 808 840 872 904 936 968 1000
9 41 73 105 137 169 201 233 265 297 329 361 393 425 457 489 521 553 585 617 649 681 713 745 777 809 841 873 905 937 969 1001
10 42 74 106 138 170 202 234 266 298 330 362 394 426 458 490 522 554 586 618 650 682 714 746 778 810 842 874 906 938 970 1002
11 43 75 107 139 171 203 235 267 299 331 363 395 427 459 491 523 555 587 619 651 683 715 747 779 811 843 875 907 939 971 1003
12 44 76 108 140 172 204 236 268 300 332 364 396 428 460 492 524 556 588 620 652 684 716 748 780 812 844 876 908 940 972 1004
13 45 77 109 141 173 205 237 269 301 333 365 397 429 461 493 525 557 589 621 653 685 717 749 781 813 845 877 909 941 973 1005
14 46 78 110 142 174 206 238 270 302 334 366 398 430 462 494 526 558 590 622 654 686 718 750 782 814 846 878 910 942 974 1006
15 47 79 111 143 175 207 239 271 303 335 367 399 431 463 495 527 559 591 623 655 687 719 751 783 815 847 879 911 943 975 1007
16 48 80 112 144 176 208 240 272 304 336 368 400 432 464 496 528 560 592 624 656 688 720 752 784 816 848 880 912 944 976 1008
17 49 81 113 145 177 209 241 273 305 337 369 401 433 465 497 529 561 593 625 657 689 721 753 785 817 849 881 913 945 977 1009
18 50 82 114 146 178 210 242 274 306 338 370 402 434 466 498 530 562 594 626 658 690 722 754 786 818 850 882 914 946 978 1010
19 51 83 115 147 179 211 243 275 307 339 371 403 435 467 499 531 563 595 627 659 691 723 755 787 819 851 883 915 947 979 1011
20 52 84 116 148 180 212 244 276 308 340 372 404 436 468 500 532 564 596 628 660 692 724 756 788 820 852 884 916 948 980 1012
21 53 85 117 149 181 213 245 277 309 341 373 405 437 469 501 533 565 597 629 661 693 725 757 789 821 853 885 917 949 981 1013
22 54 86 118 150 182 214 246 278 310 342 374 406 438 470 502 534 566 598 630 662 694 726 758 790 822 854 886 918 950 982 1014
23 55 87 119 151 183 215 247 279 311 343 375 407 439 471 503 535 567 599 631 663 695 727 759 791 823 855 887 919 951 983 1015
24 56 88 120 152 184 216 248 280 312 344 376 408 440 472 504 536 568 600 632 664 696 728 760 792 824 856 888 920 952 984 1016
25 57 89 121 153 185 217 249 281 313 345 377 409 441 473 505 537 569 601 633 665 697 729 761 793 825 857 889 921 953 985 1017
26 58 90 122 154 186 218 250 282 314 346 378 410 442 474 506 538 570 602 634 666 698 730 762 794 826 858 890 922 954 986 1018
27 59 91 123 155 187 219 251 283 315 347 379 411 443 475 507 539 571 603 635 667 699 731 763 795 827 859 891 923 955 987 1019
28 60 92 124 156 188 220 252 284 316 348 380 412 444 476 508 540 572 604 636 668 700 732 764 796 828 860 892 924 956 988 1020
29 61 93 125 157 189 221 253 285 317 349 381 413 445 477 509 541 573 605 637 669 701 733 765 797 829 861 893 925 957 989 1021
30 62 94 126 158 190 222 254 286 318 350 382 414 446 478 510 542 574 606 638 670 702 734 766 798 830 862 894 926 958 990 1022
31 63 95 127 159 191 223 255 287 319 351 383 415 447 479 511 543 575 607 639 671 703 735 767 799 831 863 895 927 959 991 1023
32 64 96 128 160 192 224 256 288 320 352 384 416 448 480 512 544 576 608 640 672 704 736 768 800 832 864 896 928 960 992 1024
[SUCCESS] Kernel validation.

C:\Users\mariana\Documents\github-mariana\parallel-computing-cuda\11102021\ex01\x64\Debug\ex01.exe (process 15120) exited with
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when
Press any key to close this window . . .
```

Figure 3: img