```
// headers
#include <stdio.h>

int main(void)
{
    // function declarations
    void PrintCUDADeviceProperties(void);

    // code
    PrintCUDADeviceProperties();
}

void PrintCUDADeviceProperties(void)
{
    // function declarations
    int ConvertSMVersionNumberToCores(int, int);

    // code
    printf("CUDA INFORMATION :\n");
    printf
      ("================================================================================
      ==\n");

    cudaError_t ret_cuda_rt;
    int dev_count;
    ret_cuda_rt = cudaGetDeviceCount(&dev_count);
    if (ret_cuda_rt != cudaSuccess)
    {
        printf("CUDA Runtime API Error - cudaGetDeviceCount() Failed Due To %s.
          Exitting Now ...\n", cudaGetErrorString(ret_cuda_rt));
    }
    else if (dev_count == 0)
    {
        printf("There Is No CUDA Supprted Device On This System. Exitting Now ...
          \n");
        return;
    }
    else
    {
        printf("Total Number Of CUDA Supporting GPU Device/Devices On This
          System : %d\n", dev_count);
        for (int i = 0; i<dev_count; i++)
        {
            cudaDeviceProp dev_prop;
            int driverVersion = 0, runtimeVersion = 0;

            ret_cuda_rt = cudaGetDeviceProperties(&dev_prop, i);
            if (ret_cuda_rt != cudaSuccess)
            {
                printf("%s in %s at line %d\n", cudaGetErrorString(ret_cuda_rt),
                  __FILE__, __LINE__);
                return;
```

```
47              }
48              printf("\n");
49              cudaDriverGetVersion(&driverVersion);
50              cudaRuntimeGetVersion(&runtimeVersion);
51              printf("******** CUDA DRIVER AND RUNTIME INFORMATION ********\n");
52              printf("=====================================================\n");
53              printf("CUDA Driver Version                                 : %d.%d
                  \n", driverVersion / 1000, (driverVersion % 100) / 10);
54              printf("CUDA Runtime Version                                : %d.%d
                  \n", runtimeVersion / 1000, (runtimeVersion % 100) / 10);
55              printf("=====================================================\n");
56              printf("********** GPU DEVICE GENERAL INFORMATION **********\n");
57              printf("=====================================================\n");
58              printf("GPU Device Number                                   : %d\n",
                   i);
59              printf("GPU Device Name                                     : %s\n",
                   dev_prop.name);
60              printf("GPU Device Compute Capability                       : %d.%d
                  \n", dev_prop.major, dev_prop.minor);
61              printf("GPU Device Clock Rate                               : %d\n",
                   dev_prop.clockRate);
62              printf("GPU Device Type                                     : ");
63              if (dev_prop.integrated)
64                  printf("Integrated ( On-Board )\n");
65              else
66                  printf("Discrete ( Card )\n");
67              printf("\n");
68              printf("********** GPU DEVICE MEMORY INFORMATION ***********\n");
69              printf("=====================================================\n");
70              printf("GPU Device Total Memory                             : %.0f
                  GB = %.0f MB = %llu Bytes\n", ((float)dev_prop.totalGlobalMem /
                  1048576.0f) / 1024.0f, (float)dev_prop.totalGlobalMem / 1048576.0f,
                   (unsigned long long) dev_prop.totalGlobalMem);
71              printf("GPU Device Available Memory                         : %lu
                  Bytes\n", (unsigned long)dev_prop.totalConstMem);
72              printf("GPU Device Host Memory Mapping Capability           : ");
73              if (dev_prop.canMapHostMemory)
74                  printf("Yes ( Can Map Host Memory To Device Memory )\n");
75              else
76                  printf("No ( Can Not Map Host Memory To Device Memory )\n");
77              printf("\n");
78              printf("****** GPU DEVICE MULTIPROCESSOR INFORMATION ********\n");
79              printf("=====================================================\n");
80              printf("GPU Device Number Of SMProcessors                   : %d\n",
                   dev_prop.multiProcessorCount);
81              printf("GPU Device Number Of Cores Per SMProcessors         : %d\n",
                   ConvertSMVersionNumberToCores(dev_prop.major, dev_prop.minor));
82              printf("GPU Device Total Number Of Cores                    : %d\n",
                   ConvertSMVersionNumberToCores(dev_prop.major, dev_prop.minor) *
                  dev_prop.multiProcessorCount);
83              printf("GPU Device Shared Memory Per SMProcessor            : %lu
                  \n", (unsigned long)dev_prop.sharedMemPerBlock);
```

```
84              printf("GPU Device Number Of Registers Per SMProcessor       : %d\n",
                   dev_prop.regsPerBlock);
85              printf("\n");
86              printf("*********** GPU DEVICE THREAD INFORMATION **********\n");
87              printf("====================================================\n");
88              printf("GPU Device Maximum Number Of Threads Per SMProcessor : %d\n",
                   dev_prop.maxThreadsPerMultiProcessor);
89              printf("GPU Device Maximum Number Of Threads Per Block       : %d\n",
                   dev_prop.maxThreadsPerBlock);
90              printf("GPU Device Threads In Warp                           : %d\n",
                   dev_prop.warpSize);
91              printf("GPU Device Maximum Thread Dimensions                 : ( %d,
                  %d, %d )\n", dev_prop.maxThreadsDim[0], dev_prop.maxThreadsDim[1],
                  dev_prop.maxThreadsDim[2]);
92              printf("GPU Device Maximum Grid Dimensions                   : ( %d,
                  %d, %d )\n", dev_prop.maxGridSize[0], dev_prop.maxGridSize[1],
                  dev_prop.maxGridSize[2]);
93              printf("\n");
94              printf("*********** GPU DEVICE DRIVER INFORMATION **********\n");
95              printf("====================================================\n");
96              printf("GPU Device has ECC support                           : %s\n",
                   dev_prop.ECCEnabled ? "Enabled" : "Disabled");
97  #if defined(WIN32) || defined(_WIN32) || defined(WIN64) || defined(_WIN64)
98              printf("GPU Device CUDA Driver Mode ( TCC Or WDDM )          : %s\n",
                   dev_prop.tccDriver ? "TCC ( Tesla Compute Cluster Driver )" :
                  "WDDM ( Windows Display Driver Model )");
99  #endif
100             printf
                  ("*************************************************************
                  **********\n");
101         }
102     }
103 }
104
105 int ConvertSMVersionNumberToCores(int major, int minor)
106 {
107     // Defines for GPU Architecture types (using the SM version to determine the
        # of cores per SM
108     typedef struct
109     {
110         int SM; // 0xMm (hexidecimal notation), M = SM Major version, and m = SM
            minor version
111         int Cores;
112     } sSMtoCores;
113
114     sSMtoCores nGpuArchCoresPerSM[] =
115     {
116         { 0x20, 32 }, // Fermi Generation (SM 2.0) GF100 class
117         { 0x21, 48 }, // Fermi Generation (SM 2.1) GF10x class
118         { 0x30, 192 }, // Kepler Generation (SM 3.0) GK10x class
119         { 0x32, 192 }, // Kepler Generation (SM 3.2) GK10x class
120         { 0x35, 192 }, // Kepler Generation (SM 3.5) GK11x class
```

```
121         { 0x37, 192 }, // Kepler Generation (SM 3.7) GK21x class
122         { 0x50, 128 }, // Maxwell Generation (SM 5.0) GM10x class
123         { 0x52, 128 }, // Maxwell Generation (SM 5.2) GM20x class
124         { 0x53, 128}, // Maxwell Generation (SM 5.3) GM20x class
125         { 0x60, 64 }, // Pascal Generation (SM 6.0) GP100 class
126         { 0x61, 128}, // Pascal Generation (SM 6.1) GP10x class
127         { 0x62, 128}, // Pascal Generation (SM 6.2) GP10x class
128         { -1, -1 }
129     };
130
131     int index = 0;
132
133     while (nGpuArchCoresPerSM[index].SM != -1)
134     {
135         if (nGpuArchCoresPerSM[index].SM == ((major << 4) + minor))
136         {
137             return nGpuArchCoresPerSM[index].Cores;
138         }
139
140         index++;
141     }
142
143     // If we don't find the values, we default use the previous one to run
        properly
144     printf("MapSMtoCores for SM %d.%d is undefined.  Default to use %d Cores/SM
        \n", major, minor, nGpuArchCoresPerSM[index - 1].Cores);
145     return(nGpuArchCoresPerSM[index - 1].Cores);
146 }
147
```