

URUCHOMIAMY CUDA NA KLASTRZE

Mirosław Najbuk m.najbuk@uwb.edu.pl

Uniwersytet w Białymstoku,

Uniwersyteckie Centrum Obliczeniowe

Białystok 2016

P1. Zasady ogólne.

1.1. Zainstalowana wersja oprogramowania CUDA.

CUDA 7.5 gcc 4.9.3 (sprzęt Tesla K80)

1.2. Lokalizacja przykładów CUDA.

```
# /home/NVIDIA*
```

1.3. Sprawdzenie dostępnych modułów

```
# module avail
```

1.4. Wgranie modułu CUDA.

```
# module load cuda
```

1.5. Sprawdzamy poprawność wgrania modułu.

```
# which nvcc
```

1.6. *Zmienne środowiskowe CUDA.

```
# export PATH=/usr/local/cuda-7.5/bin:$PATH
```

```
# export LD_LIBRARY_PATH=/usr/local/cuda-7.5/lib64:$LD_LIBRARY_PATH
```

** opcjonalnie*

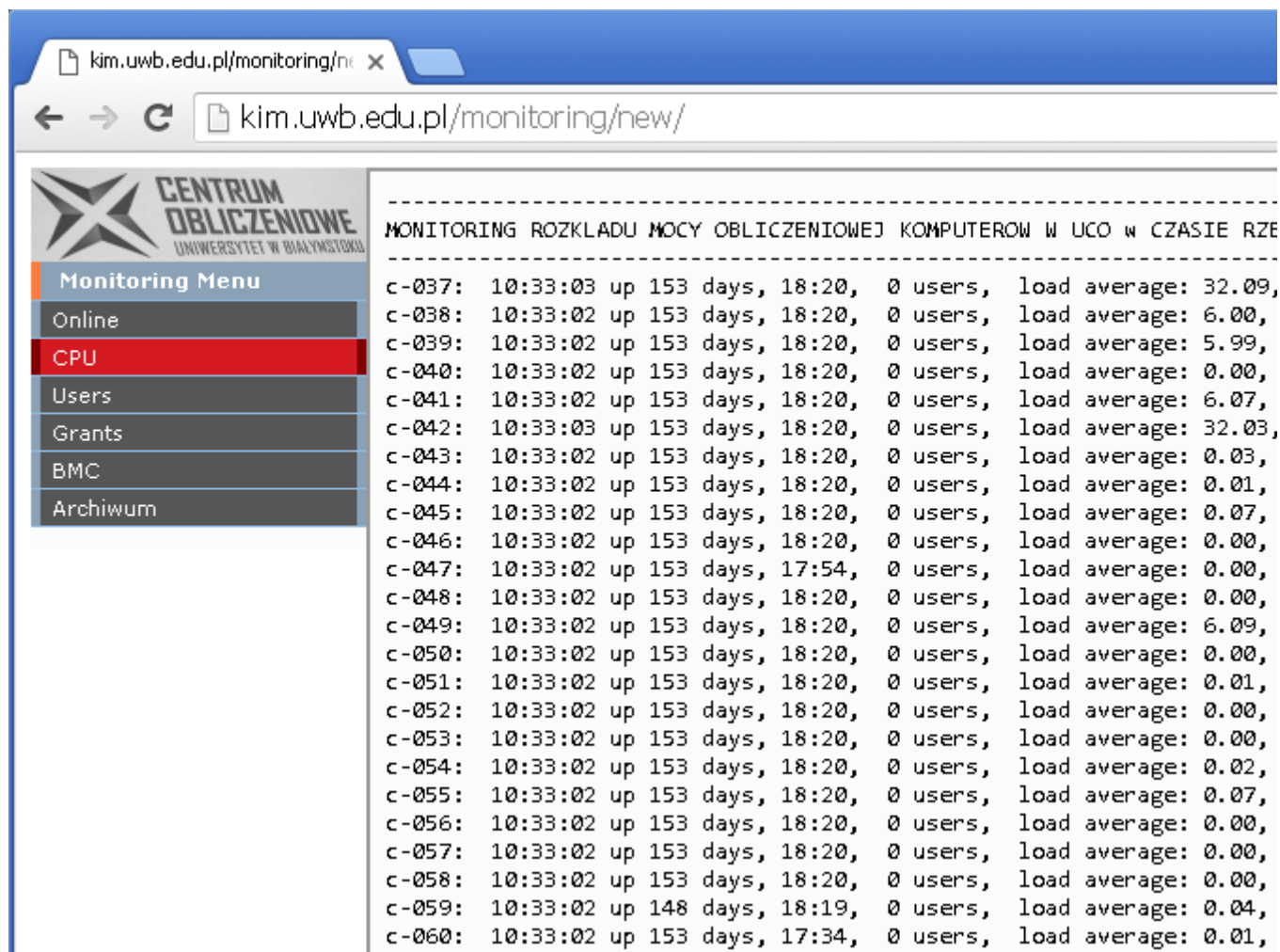
2. Uruchamianie zadania CUDA w systemie kolejkowym.

2.1. Informacje ogólne

Wykaz węzłów z dostępnością rozwiązań CUDA:

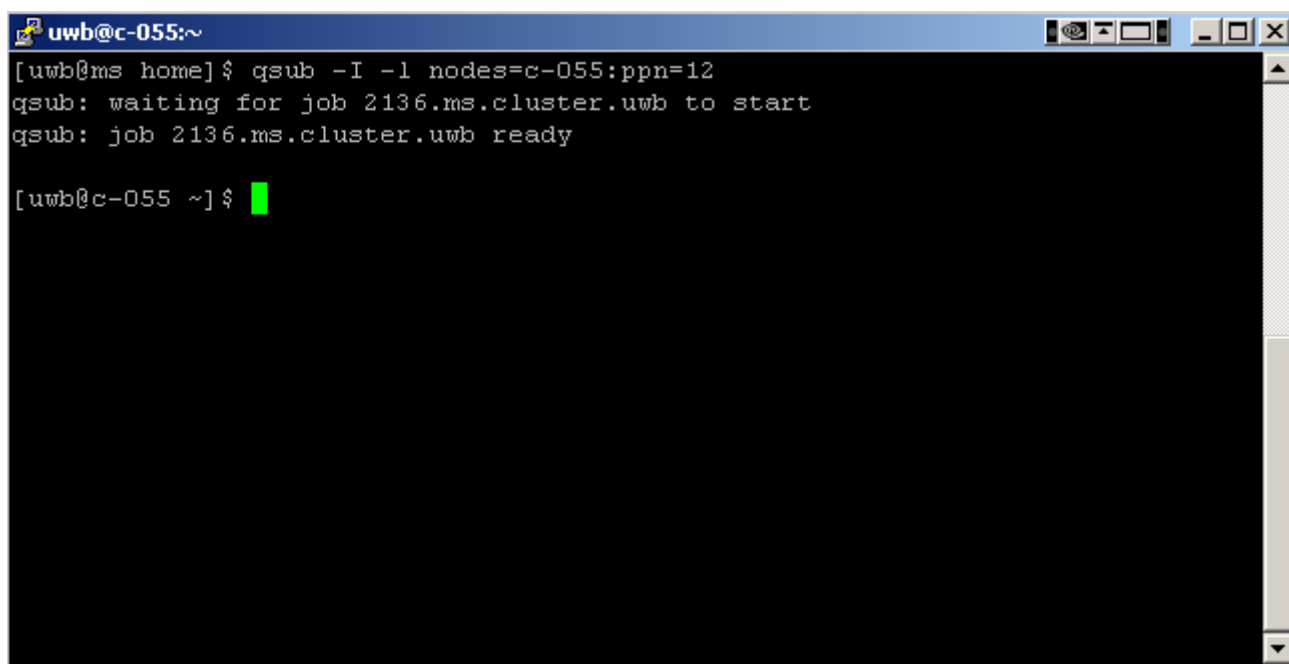
[nazwa węzła w klastrze]

- [c-037 do c-060] (**zajęcia dydaktyczne od c-055 do c-060**)
- Obciążenie węzłów CUDA w czasie rzeczywistym dostępne jest na stronie <http://kim.uwb.edu.pl/monitoring/new> (Ryc. 1.) .



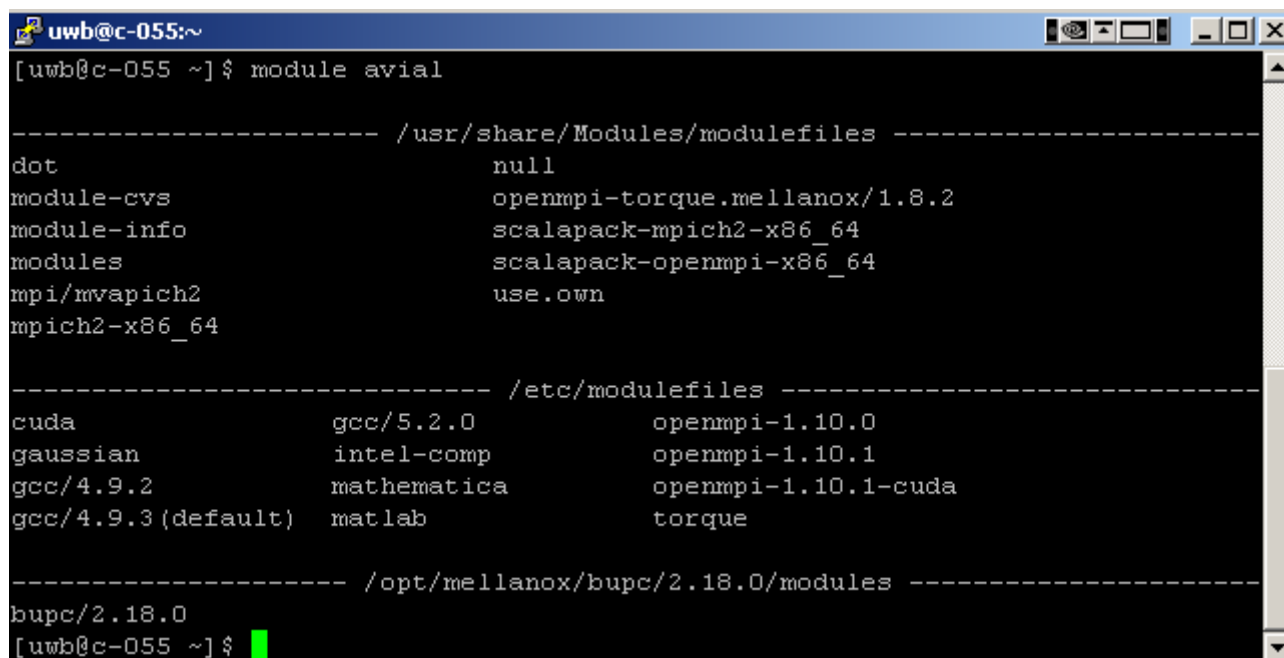
Ryc. 1. Obraz monitoringu obciążenia węzłów w UCO.

2.2. Kolejka interaktywna -I



```
uwb@c-055:~  
[uwb@ms home]$ qsub -I -l nodes=c-055:ppn=12  
qsub: waiting for job 2136.ms.cluster.uwb to start  
qsub: job 2136.ms.cluster.uwb ready  
  
[uwb@c-055 ~]$
```

Ryc. 2. Przedstawiono przykładowe polecenie do uruchomienia sesji interaktywnej z wykorzystaniem węzła c-055 i ilości rdzeni 12.



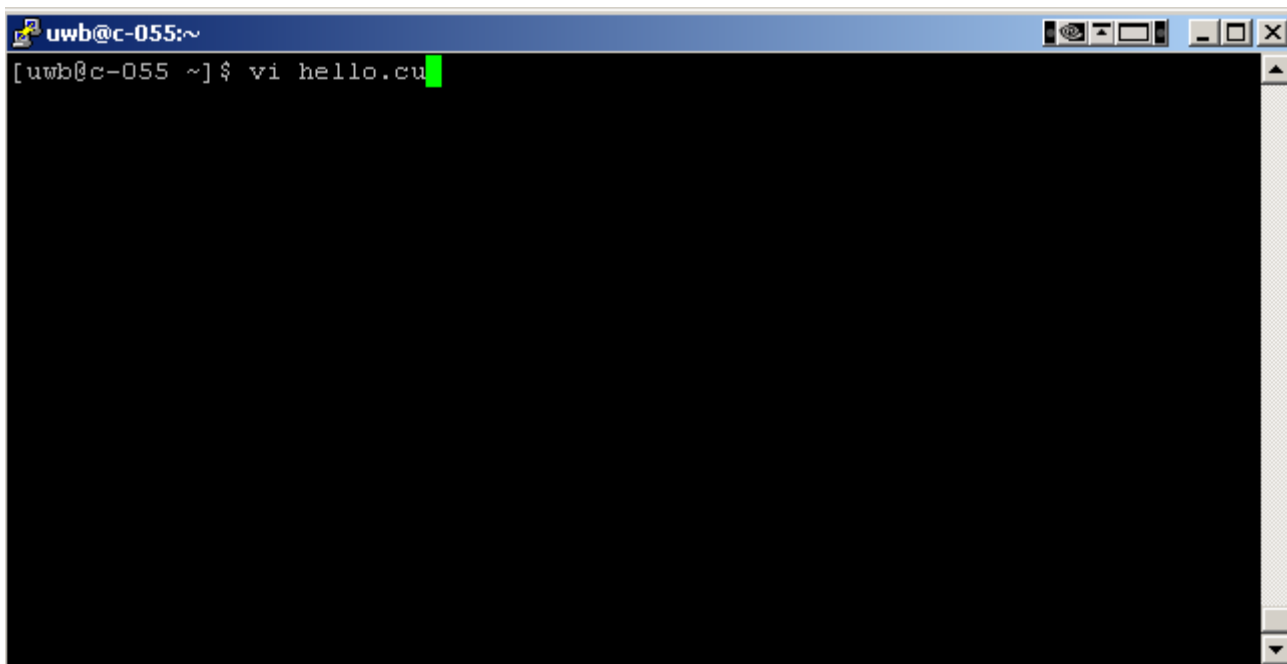
```
uwb@c-055:~  
[uwb@c-055 ~]$ module avial  
  
----- /usr/share/Modules/modulefiles -----  
dot null  
module-cvs openmpi-torque.mellanox/1.8.2  
module-info scalapack-mpich2-x86_64  
modules scalapack-openmpi-x86_64  
mpi/mvapich2 use.own  
mpich2-x86_64  
  
----- /etc/modulefiles -----  
cuda gcc/5.2.0 openmpi-1.10.0  
gaussian intel-comp openmpi-1.10.1  
gcc/4.9.2 mathematica openmpi-1.10.1-cuda  
gcc/4.9.3 (default) matlab torque  
  
----- /opt/mellanox/bupc/2.18.0/modules -----  
bupc/2.18.0  
[uwb@c-055 ~]$
```

Ryc. 3. Przedstawiono polecenie do wylistowania dostępnych modułów.



```
uwb@c-055:~  
[uwb@c-055 ~]$ module load cuda  
[uwb@c-055 ~]$ nvcc
```

Ryc. 4. Przedstawiono polecenie do załadowania zmiennych środowiskowych CUDA (module load cuda).



```
uwb@c-055:~  
[uwb@c-055 ~]$ vi hello.cu
```

Ryc. 5. Przedstawiono przykładowy program vi do redakcji plików.

```

uwb@c-055:~
// It takes the string "Hello ", prints it, then passes it to CUDA with an ar
ray
// of offsets. Then the offsets are added in parallel to produce the string "
World!"
// By Ingemar Ragnemalm 2010

#include <stdio.h>

const int N = 7;
const int blocksize = 7;

__global__
void hello(char *a, int *b)
{
    a[threadIdx.x] += b[threadIdx.x];
}

int main()
{
    char a[N] = "Hello ";
    ichar *ad; {15, 10, 6, 0, -11, 1, 0};
    int *bd;
    const int csize = N*sizeof(char);
    const int isize = N*sizeof(int);

    printf("%s", a);

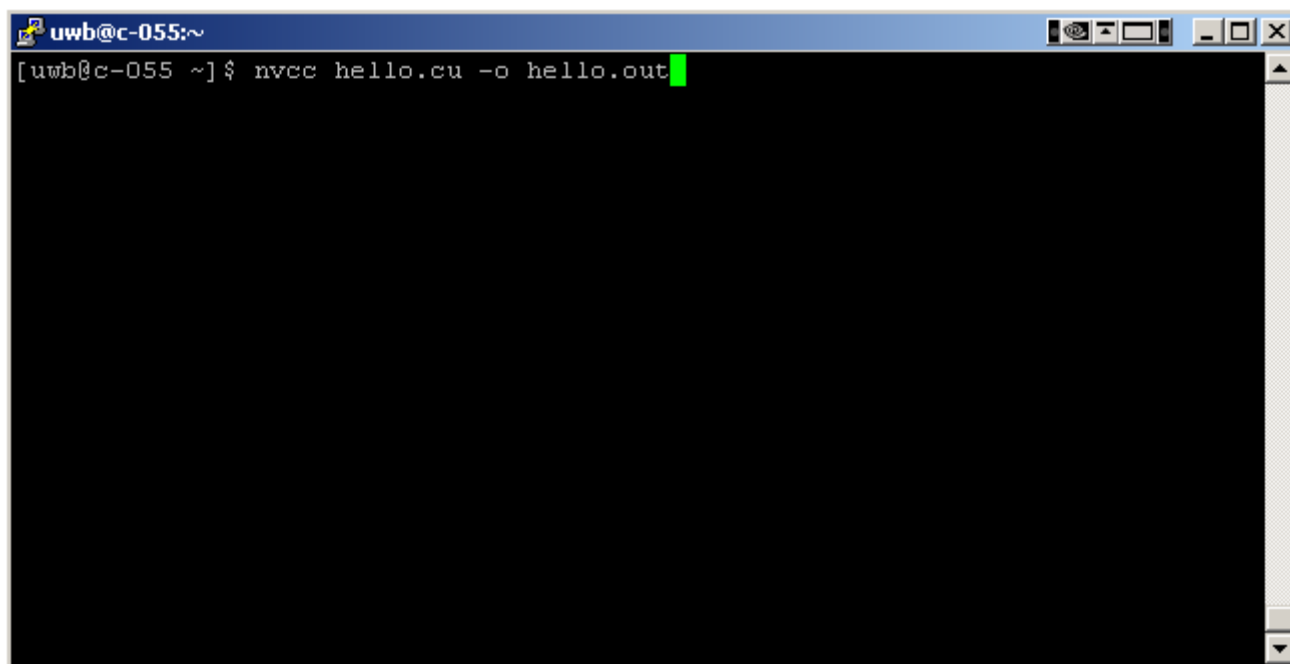
    cudaMalloc( (void**)&ad, csize );
    cudaMalloc( (void**)&bd, isize );
    cudaMemcpy( ad, a, csize, cudaMemcpyHostToDevice );
    cudaMemcpy( bd, b, isize, cudaMemcpyHostToDevice );

    dim3 dimBlock( blocksize, 1 );
    dim3 dimGrid( 1, 1 );
    hello<<<dimGrid, dimBlock>>>(ad, bd);
    cudaMemcpy( a, ad, csize, cudaMemcpyDeviceToHost );
    cudaFree( ad );

    printf("%s\n", a);
    return EXIT_SUCCESS;
}

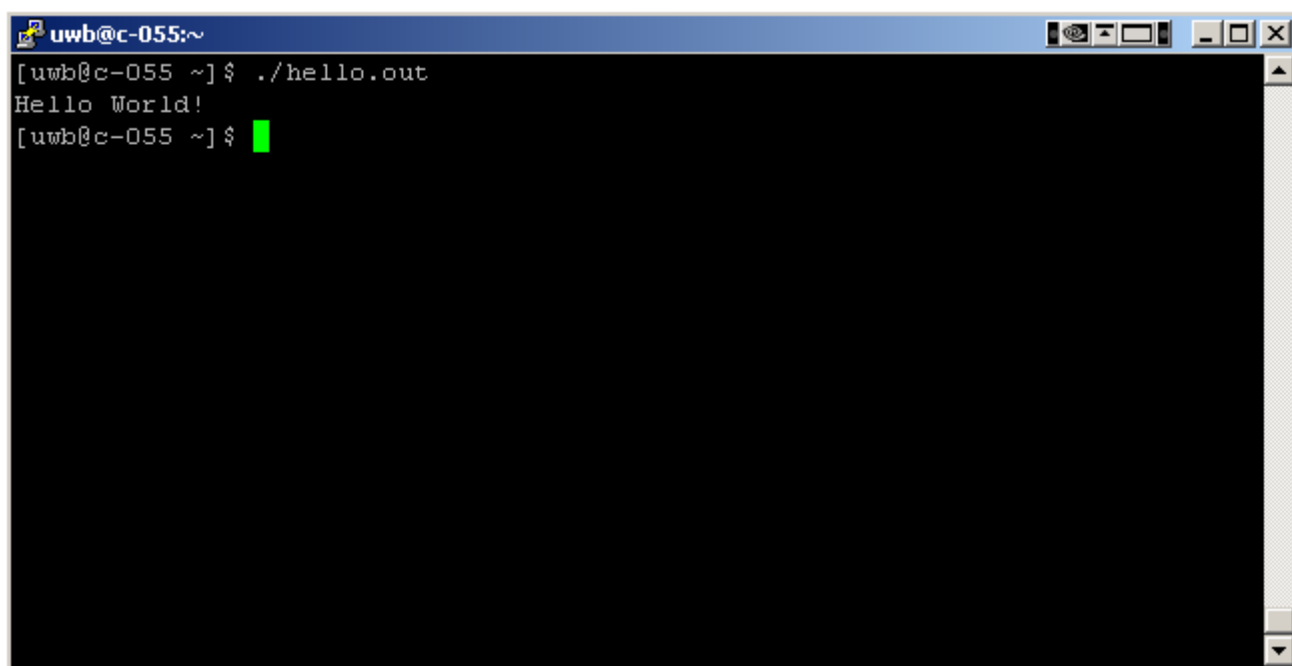
```

Ryc. 6. Przedstawiono przykład programu CUDA.



```
uwb@c-055:~  
[uwb@c-055 ~]$ nvcc hello.cu -o hello.out
```

Ryc. 7. Przedstawiono przykład kompilacji programu CUDA.



```
uwb@c-055:~  
[uwb@c-055 ~]$ ./hello.out  
Hello World!  
[uwb@c-055 ~]$
```

Ryc. 8. Wynik kompilacji programu CUDA.

```

mc [uwb@c-055]:/home/NVIDIA_CUDA-7.5_Samples-gcc-4.9.3/1_Uutilities/deviceQuery
Device 1: "Tesla K80"
  CUDA Driver Version / Runtime Version      7.5 / 7.5
  CUDA Capability Major/Minor version number: 3.7
  Total amount of global memory:             11520 MBytes (12079136768 bytes)
  (13) Multiprocessors, (192) CUDA Cores/MP: 2496 CUDA Cores
  GPU Max Clock rate:                        824 MHz (0.82 GHz)
  Memory Clock rate:                         2505 Mhz
  Memory Bus Width:                          384-bit
  L2 Cache Size:                             1572864 bytes
  Maximum Texture Dimension Size (x,y,z)    1D=(65536), 2D=(65536, 65536),
  Maximum Layered 1D Texture Size, (num) layers 1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers 2D=(16384, 16384), 2048 layers
  Total amount of constant memory:           65536 bytes
  Total amount of shared memory per block:   49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                 32
  Maximum number of threads per multiprocessor: 2048
  Maximum number of threads per block:      1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                     2147483647 bytes
  Texture alignment:                         512 bytes
  Concurrent copy and kernel execution:     Yes with 2 copy engine(s)
  Run time limit on kernels:                 No
  Integrated GPU sharing Host Memory:        No
  Support host page-locked memory mapping:   Yes
  Alignment requirement for Surfaces:        Yes
  Device has ECC support:                    Enabled
  Device supports Unified Addressing (UVA):  Yes
  Device PCI Domain ID / Bus ID / location ID: 0 / 133 / 0
  Compute Mode:
    < Default (multiple host threads can use ::cudaSetDevice() with device simul
> Peer access from Tesla K80 (GPU0) -> Tesla K80 (GPU1) : Yes
> Peer access from Tesla K80 (GPU1) -> Tesla K80 (GPU0) : Yes

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 7.5, CUDA Runtime Versi
K80
Result = PASS

[uwb@c-055 deviceQuery]$ █

```

Ryc. 9. Przedawniono wynik działania programu deviceQuery.