```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <iostream>
#include <vector>
/* ----------------------------------------------------------------
Se da o matrice ce contine N puncte 3-dimesionale (deci o matrice cu
N linii si 3 coloane). Implementati un program CUDA ce calculeaza distanta
intre un punct 3-dimensional p0 dat si fiecare din cele N puncte din matrice.

Urmati, pe rand, pasii de mai jos.

Indicatie: distanta intre 2 vectori 3-dimensionali a si b, se calculeaza astfel:

d = sqrt( (a_0 - b_0)*(a_0 - b_0) + (a_1 - b_1)*(a_1 - b_1) + (a_2 - b_2)*(a_2 - b_2) );

Unde, a_i si b_i reprezinta componenta i a vectorului a respectiv b.
----------------------------------------------------------------*/
__global__ void compute_distance(float *p, float p0_x, float p0_y, float p0_z, float *distante,
int N
);
int main()
{
  // Numarul N de puncte
  int N = 1000;
  // Matricea ce contine punctele
  float *p_h;
  // punctul p0
  float p0_x = 0;
  float p0_y = 0;
  float p0_z = 0;
  // Distantele intre p0 si fiecare p_i
  float *distante_h;
  /* ----------------------------------------------------------------
  1. Alocati memorie pe host pentru matricea p_h si vectorul distante_h
  dupa cum urmeaza:
  * Matricea este de N linii si 3 coloane si este alocată ca un singur
  bloc de memorie (o singura alocare de N*3 elemente)
  * Vectorul ce contine distantele are N elemente
  ----------------------------------------------------------------*/
  /* ------------------------ REZOLVARE PCT 1 ----------------------------*/
  /* ------------------------ CODUL TAU DE AICI ----------------------------*/


  /* ------------------------ CODUL TAU PANA AICI ----------------------------*/
  /* ----------------------------------------------------------------
  2. Initializati matricea ce contine punctele dupa cum urmeaza:
  * Primul punct (adica prima linie din matrice) are coordonatele 0,0,0,
  al 2-lea punct are 1,1,1, al 3-lea 2,2,2, etc
  ----------------------------------------------------------------*/
  /* ------------------------ REZOLVARE PCT 2 ----------------------------*/
  /* ------------------------ CODUL TAU DE AICI ----------------------------*/


  /* ------------------------ CODUL TAU PANA AICI ----------------------------*/
  // Matricea si cei doi vectori pentru memoria device
  float *p_d;
  float *distante_d;
  /* ----------------------------------------------------------------
  3. Alocati memorie pe device pentru matrice si vectorul ce contine distantele
  ----------------------------------------------------------------*/
  /* ------------------------ REZOLVARE PCT 3 ----------------------------*/
  /* ------------------------ CODUL TAU DE AICI ----------------------------*/


  /* ------------------------ CODUL TAU PANA AICI ----------------------------*/
  /* ----------------------------------------------------------------
  4. Copiati continutul matricei si vectorului b de pe host pe device
  ----------------------------------------------------------------*/
  /* ------------------------ REZOLVARE PCT 4 ----------------------------*/
```

```
  /* ---------------------- CODUL TAU DE AICI --------------------------*/


  /* ---------------------- CODUL TAU PANA AICI -------------------------*/
  // Numarul de thread-uri pe bloc
  dim3 threadsPerBlock;
  threadsPerBlock.x = 128;
  threadsPerBlock.y = 1;
  threadsPerBlock.z = 1;
  // Numarul de blocuri
  dim3 numBlocks;
  /* ---------------------------------------------------------------------
  5. Calculati numarul de blocuri astfel incat:
  * se obtine un grid 1D de thread-uri
  * Numărul de thread-uri lansate in executie exte mai mare sau egal cu N
  ---------------------------------------------------------------------*/
  /* ---------------------- REZOLVARE PCT 5 ----------------------------*/
  /* ---------------------- CODUL TAU DE AICI --------------------------*/


  /* ---------------------- CODUL TAU PANA AICI -------------------------*/
  // Se lanseaza in executie kernel-ul cuda
  compute_distance << <numBlocks, threadsPerBlock >> >(p_d, p0_x, p0_y, p0_z, distante_d, N);
  /* ---------------------------------------------------------------------
  6. Copiati continutul vectorului ce contine distantele calculate de pe device pe host
  ---------------------------------------------------------------------*/
  /* ---------------------- REZOLVARE PCT 6 ----------------------------*/
  /* ---------------------- CODUL TAU DE AICI --------------------------*/


  /* ---------------------- CODUL TAU PANA AICI -------------------------*/
  return 0;
}
__global__ void compute_distance(float *p, float p0_x, float p0_y, float p0_z, float
*distances, int
  N)
{
  // Indicele fiecarui thread
  int index;
  /* ---------------------------------------------------------------------
  7. Calculati valoarea indicelui utilizand variabilele predefinite blockDim
  blocIdx si threadIdx astfel incat: thread-ul zero are index 0, thread-ul 1
  are indicele 1, thread-ul 2 are indicele 2, etc.
  ---------------------------------------------------------------------*/
  /* ---------------------- REZOLVARE PCT 7 ----------------------------*/
  /* ---------------------- CODUL TAU DE AICI --------------------------*/


  /* ---------------------- CODUL TAU PANA AICI -------------------------*/
  /* ---------------------------------------------------------------------
  8. Calculti distanta intre punctul p0 si fiecare punct din matricea p
  ---------------------------------------------------------------------*/
  /* ---------------------- REZOLVARE PCT 8 ----------------------------*/
  /* ---------------------- CODUL TAU DE AICI --------------------------*/


  /* ---------------------- CODUL TAU PANA AICI -------------------------*/
}
```