Jacobi1d ccode:

```
#include <stdio.h>
#include <malloc.h>
#include <math.h>
#define  M      32
#define  dt     0.01
#define  dx     0.1
#define  Epsilon 0.00001
//
============================
void Inputdata(float *T){
  int i;
  for (  i = 0 ; i < M ; i++ )
     *(T+i) = 25.0;
}
//
============================
void Jacobi(float *Tnew, float *Told){
  //Tnew=T at step k+1, Told=T at step k,
  int i;
  float l,r;
  for ( i = 0 ; i < M ; i++ ) {
    l = (i==0)   ? 100.0 : *(Told+(i-1));
    r = (i==M-1) ? 25.0  : *(Told+(i+1));
    *(Tnew+i) = (l+r)/2;
  }
}
//
============================
float MaxDiff(float *Tnew, float *Told){
  int i;
  float max = 0;
  for ( i = 0 ; i < M ; i++ ) {
    if (fabs(*(Tnew+i)-*(Told+i)) > max)
      max  = fabs(*(Tnew+i)-*(Told+i));
  }
  return max;
}
//
============================
int main(){
  int i, iteration = 0;
  float *Told,*Tnew;
  Told = (float *) malloc
((M)*sizeof(float));
  Tnew = (float *) malloc
((M)*sizeof(float));
  Inputdata(Told);
  float diff;
  do {
    iteration++;
    Jacobi(Tnew,Told);
    diff = MaxDiff(Tnew,Told);
    // Update value
    for ( i = 0 ; i < M ; i++ )
      *(Told+i) = *(Tnew+i);
  } while  (diff > Epsilon);
  //
  printf("Number of Iteration: %d \n",
iteration);
```

```
  for ( i = 0 ; i < M ; i++ ) printf("%f \
n",*(Tnew+i));
  return 0;
}

cuda
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cuda_runtime.h>

#define M       32
#define dt      0.01
#define dx      0.1
#define Epsilon 0.00001
#define BLOCK_SIZE 256

__global__ void jacobi_kernel(float *Tnew,
float *Told, int size) {
    int i = blockIdx.x * blockDim.x +
threadIdx.x;

    if (i < size) {
        float l, r;
        l = (i == 0) ? 100.0f : Told[i-1];
        r = (i == size-1) ? 25.0f : Told[i+1];
        Tnew[i] = (l + r) / 2.0f;
    }
}
__global__ void max_diff_kernel(float
*Tnew, float *Told, float *diff_array, int
size) {
    int i = blockIdx.x * blockDim.x +
threadIdx.x;
    int tid = threadIdx.x;
    __shared__ float sdata[BLOCK_SIZE];
    sdata[tid] = 0.0f;

    if (i < size) {
        sdata[tid] = fabsf(Tnew[i] - Told[i]);
    }
    __syncthreads();

    for (int s = blockDim.x / 2; s > 0; s
>>= 1) {
        if (tid < s && (i + s) < size) {
            if (sdata[tid + s] > sdata[tid]) {
                sdata[tid] = sdata[tid + s];
            }
        }
        __syncthreads();
    }
    if (tid == 0) {
        diff_array[blockIdx.x] = sdata[0];
    }
}
void inputdata(float *T, int size) {
    for (int i = 0; i < size; i++) {
        T[i] = 25.0f;
    }
```

```
}


float find_max_cpu(float *array, int size)
{
    float max_val = 0.0f;
    for (int i = 0; i < size; i++) {
        if (array[i] > max_val) {
            max_val = array[i];
        }
    }
    return max_val;
}
#define CUDA_CHECK(call) \
    do { \
        cudaError_t error = call; \
        if (error != cudaSuccess) { \
            fprintf(stderr, "CUDA error at %s:
%d - %s\n", __FILE__, __LINE__, \
                    cudaGetErrorString(error)); \
            exit(1); \
        } \
    } while(0)


int main() {
    int iteration = 0;
    float *h_Told, *h_Tnew;
    float *d_Told, *d_Tnew, *d_diff_array;
    float *h_diff_array;


    int grid_size = (M + BLOCK_SIZE - 1) /
BLOCK_SIZE;

    h_Told = (float*)malloc(M *
sizeof(float));
    h_Tnew = (float*)malloc(M *
sizeof(float));
    h_diff_array = (float*)malloc(grid_size
* sizeof(float));


    inputdata(h_Told, M);
    CUDA_CHECK(cudaMalloc(&d_Told, M *
sizeof(float)));
    CUDA_CHECK(cudaMalloc(&d_Tnew, M
* sizeof(float)));

CUDA_CHECK(cudaMalloc(&d_diff_array,
grid_size * sizeof(float)));


    CUDA_CHECK(cudaMemcpy(d_Told,
h_Told, M * sizeof(float),
cudaMemcpyHostToDevice));


    float diff;
    do {
        iteration++;
        jacobi_kernel<<<grid_size,
BLOCK_SIZE>>>(d_Tnew, d_Told, M);
        CUDA_CHECK(cudaGetLastError());


        max_diff_kernel<<<grid_size,
BLOCK_SIZE>>>(d_Tnew, d_Told,
d_diff_array, M);
        CUDA_CHECK(cudaGetLastError());
```