```
#include <stdio.h>
#include <unistd.h>
#include <pwd.h>

#define MAX_THREADS 256

#define gflops(n,ms) (((n*18.0)/(ms/1.0e+3))/1.0e+9)

__device__ double f(double x)
{
   double temp;
   temp = x*x*x+1;
   return 9*x/(temp*temp);
}

__global__ void area_kernel(double *local_area, long N, double a, double b)
{
   double dx, x;
   long i = blockDim.x*blockIdx.x+threadIdx.x;
   int half;
   extern __shared__ double sdata[];

   dx = (b-a)/(double)N;
   x = a + (double)i*dx;
   if (i < N)
      sdata[threadIdx.x] = 0.5*(f(x)+f(x+dx))*dx;
   else
      sdata[threadIdx.x] = 0.0;

   // do reduce in shared memory
   half = 1<<(int)(log2((float)(blockDim.x-1)));
   for(unsigned int s=half; s>0; s>>=1)
   {
      if (threadIdx.x+s < blockDim.x)
         if (threadIdx.x < s) sdata[threadIdx.x] += sdata[threadIdx.x+s];
      __syncthreads();
   }

   // write result for this block to global memory
   if (threadIdx.x == 0) local_area[blockIdx.x] = sdata[0];
}

int main(int argc, char *argv[])
{
   double *local_area, *local_area_d, area, a, b;
   long N;
   int nt, smsize, dev, i;
   cudaEvent_t start, stop;
   float elapsed;
   cudaDeviceProp deviceProp;

   if (argc != 3) {
      fprintf(stderr, "usage: %s #segments #threads\n", argv[0]);
      exit(1);
   }
   N = atol(argv[1]);
   nt = atoi(argv[2]);

   if (nt > MAX_THREADS) {
      nt = MAX_THREADS;
      fprintf(stderr, "%d threads are used.\n", MAX_THREADS);
   }

   dim3 dimBlock(nt);
   dim3 dimGrid((N+dimBlock.x-1)/dimBlock.x);

   smsize = sizeof(double)*nt;

   dev = (getpwuid(getuid())->pw_name[3]-’0’)%2? 1: 0;
   cudaSetDevice(dev);
   cudaGetDeviceProperties(&deviceProp, dev);
   printf("Device(%d) used: \"%s\"\n", dev, deviceProp.name);

   cudaEventCreate(&start);
   cudaEventCreate(&stop);
   cudaEventRecord(start, 0);

   local_area = (double*)malloc(sizeof(double)*dimGrid.x);
   cudaMalloc((void **)&local_area_d, sizeof(double)*dimGrid.x);

   a = 0.0, b = 2.0;
   area_kernel<<<dimGrid, dimBlock, smsize>>>(local_area_d, N, a, b);

   // copy values from GPU memory to CPU memory
   cudaMemcpy(local_area, local_area_d, sizeof(double)*dimGrid.x, cudaMemcpyDeviceToHo
st);

   cudaEventRecord(stop, 0);
   cudaEventSynchronize(stop);
   cudaEventElapsedTime(&elapsed, start, stop);

   // sum local_areas computed on GPU
   area = 0.0;
   for (i=0; i<dimGrid.x; i++)
      area += local_area[i];

   printf("area: %5.5lf\n", area);
   printf("elapsed time: %5.2f milliseconds", elapsed);
   printf(" (GFLOPS: %5.2f)\n", (N*18.0/(elapsed/1.0e+3))/1.0e+9);

   free(local_area);
   cudaFree(local_area_d);

   exit(0);
}
```