ARQUITECTURA DE COMPUTADORES

1.CÓDIGO

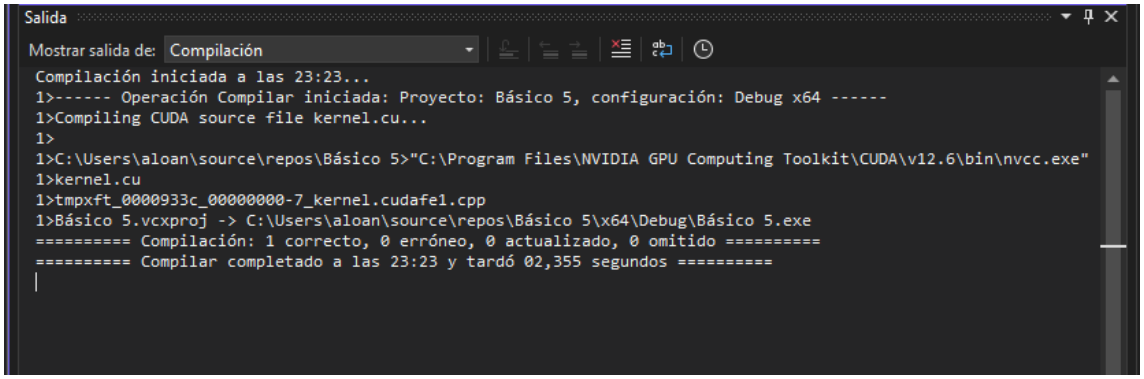
```
/*
* ARQUITECTURA DE COMPUTADORES
* 2º Grado en Ingenieria Informatica
*
* ENTREGA Básico 5:
* >> Cálculo de π mediante reducción paralela
*
* Alumno: Antonio Alonso Briones
* Fecha: 16/10/2024
*
*/

// includes
#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime.h>
#include <math.h>
#include <device_launch_parameters.h>
// Función para verificar si un número es potencia de 2
int esPotenciaDeDos(int x)
{
    return (x != 0) && ((x & (x - 1)) == 0);
}

// Kernel para calcular π utilizando reducción paralela
__global__ void calcular_pi(float* datos, float* resultado, int N)
{
    int myID = threadIdx.x;

    // Cada hilo calcula su término de la serie
    if (myID < N)
    {
        float n = (float)(myID + 1);
        datos[myID] = 1.0f / (n * n);
    }

    // Sincronización de hilos
    __syncthreads();

    // Reducción paralela
    int salto = N / 2;
    while (salto > 0)
    {
        if (myID < salto)
        {
            datos[myID] += datos[myID + salto];
        }
        // Sincronización de hilos
        __syncthreads();
        salto /= 2;
    }

    // El hilo 0 escribe el resultado final
    if (myID == 0)
    {
        *resultado = datos[0];
```

```
    }
}

int main(int argc, char** argv)
{
    int N;
    printf("Introduce el número de términos (potencia de 2): ");
    scanf("%d", &N);

    // Verificar que N sea potencia de 2
    if (!esPotenciaDeDos(N))
    {
        printf("El número de términos debe ser una potencia de 2.\n");
        return 1;
    }

    // Obtener las propiedades del dispositivo
    cudaDeviceProp deviceProp;
    cudaGetDeviceProperties(&deviceProp, 0);

    // Cálculo del número total de núcleos (SP)
    int cudaCores;
    int SM = deviceProp.multiProcessorCount;
    int major = deviceProp.major;
    int minor = deviceProp.minor;
    const char* archName;
    switch (major)
    {
    case 1: // Tesla
        archName = "TESLA";
        cudaCores = 8;
        break;
    case 2: // Fermi
        archName = "FERMI";
        if (minor == 0)
            cudaCores = 32;
        else
            cudaCores = 48;
        break;
    case 3: // Kepler
        archName = "KEPLER";
        cudaCores = 192;
        break;
    case 5: // Maxwell
        archName = "MAXWELL";
        cudaCores = 128;
        break;
    case 6: // Pascal
        archName = "PASCAL";
        if (minor == 1 || minor == 2)
            cudaCores = 128;
        else
            cudaCores = 64;
        break;
    case 7: // Volta y Turing
        if (minor == 0)
            archName = "VOLTA";
        else
            archName = "TURING";
        cudaCores = 64;
        break;
    case 8: // Ampere
        archName = "AMPERE";
```

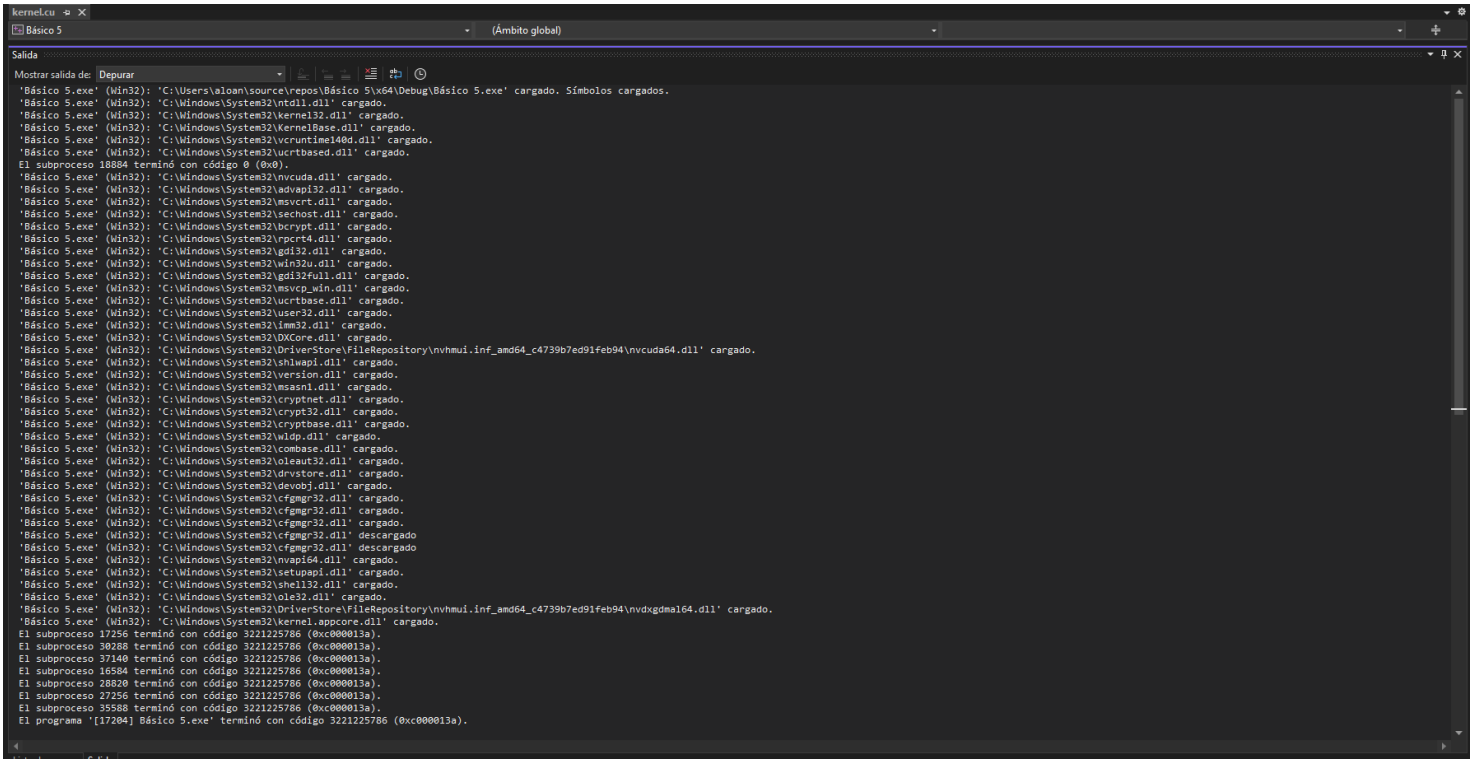
```
            cudaCores = 64;
            break;
        case 9: // Hopper
            archName = "HOPPER";
            cudaCores = 128;
            break;
        default: // Arquitectura desconocida
            archName = "DESCONOCIDA";
            cudaCores = 0;
            break;
        }
        int totalCores = SM * cudaCores;

        // Mostrar las propiedades del dispositivo
        printf("\n> Propiedades del dispositivo seleccionado:\n");
        printf("Nombre: %s\n", deviceProp.name);
        printf("Arquitectura CUDA: %s\n", archName);
        printf("Capacidad de cómputo: %d.%d\n", major, minor);
        printf("Número de multiprocesadores: %d\n", SM);
        printf("Número de núcleos CUDA (%d x %d): %d\n", SM, cudaCores, totalCo-
res);
        printf("Máximo número de hilos por bloque: %d\n", deviceProp.maxTh-
readsPerBlock);

        // Verificar que N no exceda el máximo de hilos por bloque
        if (N > deviceProp.maxThreadsPerBlock)
        {
            printf("El número de términos (%d) excede el máximo número de hilos
por bloque (%d).\n", N, deviceProp.maxThreadsPerBlock);
            // Ajustar N al máximo permitido y que sea potencia de 2
            int temp = deviceProp.maxThreadsPerBlock;
            while (!esPotenciaDeDos(temp))
            {
                temp--;
            }
            N = temp;
            printf("Ajustando N a %d para ser potencia de 2.\n", N);
        }

        printf("Número de hilos lanzados: %d\n", N);

        // Declaración de punteros para host y device
        float* hst_datos;
        float* dev_datos;
        float hst_resultado;
        float* dev_resultado;

        // Reserva de memoria en el host
        hst_datos = (float*)malloc(N * sizeof(float));

        // Reserva de memoria en el device
        cudaMalloc((void**)&dev_datos, N * sizeof(float));
        cudaMalloc((void**)&dev_resultado, sizeof(float));

        // Lanzamiento del kernel
        printf("\n> Calculando...\n");
        calcular_pi << <1, N >> > (dev_datos, dev_resultado, N);

        // Copiar el resultado al host
        cudaMemcpy(&hst_resultado, dev_resultado, sizeof(float), cudaMemcpyDevi-
ceToHost);

        // Calcular la aproximación de π
```

```
    float pi_aprox = sqrtf(6.0f * hst_resultado);

    // Calcular el error relativo
    float pi_ref = 3.141593f;
    float error = fabsf(pi_ref - pi_aprox) / pi_ref * 100.0f;

    // Mostrar resultados
    printf("\n> RESULTADOS:\n");
      printf("Valor de π: %.6f\n", pi_ref); // Valor de π conocido (6 deci-
males)
    printf("Aproximación de π: %.6f\n", pi_aprox);
    printf("Error relativo: %.6f%%\n", error);

    // Liberar memoria
    free(hst_datos);
    cudaFree(dev_datos);
    cudaFree(dev_resultado);

    // Salida
    printf("*************************************************\n");
    printf("<pulsa [INTRO] para finalizar>");
    getchar();
    getchar();
    return 0;
}
```

2. RESULTADO DE LA COMPILACIÓN



3. RESULTADO DE LA DEPURACIÓN



4. SALIDA POR PANTALLA