

Práctica 08a.

Cálculo del valor E (irracional)



U.A.Q. Fac. de Informática
Dra. Sandra Luz Canchola Magdaleno

Correo: sandra.canchola@uaq.mx

Dra. Reyna Moreno Beltrán

Correo: reyna.moreno@uaq.mx



Valor de Constante de Erdős–Borwein

La **Constante Erdős–Borwein** es la suma del [recíproco](#) de los [números de Mersenne](#). Se le llamó así en referencia a [Paul Erdős](#) y [Peter Borwein](#).

Se define como:

$$E = \sum_{n=1}^{\infty} \frac{1}{2^n - 1} \approx 1,606695152415291763 \dots$$

Un **número de Mersenne** es un número entero positivo M que es una unidad menor que una potencia entera positiva de 2:

$$M_n = 2^n - 1.$$

$$E = \underbrace{\frac{1}{2^1 - 1} + \frac{1}{2^2 - 1} + \frac{1}{2^3 - 1} + \frac{1}{2^4 - 1} + \frac{1}{2^5 - 1} + \frac{1}{2^6 - 1} + \dots + \frac{1}{2^n - 1}}_{n \text{ términos}}$$

Términos (n)	Valor de E
10	1.605718271890745896257612
100	1.606695152415291261149832
1,000	1.606695152415291261149832
10,000	1.606695152415291261149832

Proyecto CUDA



Create a new project

Choose a project template with code scaffolding to get started



CUDA 11.7 Runtime

A project that uses the CUDA 11.7 runtime

C++

CUDA

Windows

Linux

Cloud

Console

DataScience

Desktop

Machine Learning



CUDA 12.1 Runtime

A project that uses the CUDA 12.1 runtime

C++

CUDA

Windows

Linux

Cloud

Console

DataScience

Desktop

Machine Learning

Proyecto CUDA

Configure your new project

CUDA 12.1 Runtime

C++

CUDA

Windows

Linux

Cloud

Console

DataScience

Desktop

Machine Learning

Project name

Prog08_CalculoPI

Location

C:\TrabajoLaboratorio\CUDATopico2\Projects\Verano2024

...

Solution name ⓘ

Prog08_CalculoPI

☐ Place solution and project in the same directory

Project will be created in "C:\TrabajoLaboratorio\CUDATopico2\Projects\Verano2024
\Prog08_CalculoPI\Prog08_CalculoPI"

Back

Create

Operaciones de memoria (CPU)

- **malloc.**- Reserva un bloque de memoria de un tamaño definido de bytes, retornando un apuntador al inicio de dicho bloque. El contenido de dicho bloque no se inicializa por lo que es indeterminado. Ejemplo:

```
void* malloc (size_t size);  
buffer = (char*) malloc (sizeof(char)*100);
```

- **memset.**- asigna valores en secciones de memoria. Ejemplo:
Memset(variable, valor_a_asignar, tamaño_de_memoria)
Donde: tamaño_de_memoria se define como n * sizeof(tipo)

- **memcpy.**- Copia el contenido de un bloque de memoria referenciado por un apuntador a otro apuntador. Ejemplo:

```
void* memcpy( void* dest, const void* src, std::size_t count );  
memcpy (ptrDest, ptrOrigen, sizeof(int)*100);
```

- **free.**- Liberar la memoria reservada con el comando malloc. Ejemplo:
free(pointerName);
free(array2);

Operaciones de memoria (GPU)

- **cudaMalloc**.- asigna una sección de memoria en GPU de acuerdo con el espacio solicitado.

Ejemplo:

```
cudaMalloc((void**) &apuntador, tamaño_de_memoria)
```

Donde: tamaño_de_memoria se define como $n * \text{sizeof}(\text{tipo})$

- **cudaMemset**.- asigna valores en secciones de memoria.

Ejemplo:

```
Memset(apuntador, valor_a_asignar, tamaño_de_memoria)
```

Donde: tamaño_de_memoria se define como $n * \text{sizeof}(\text{tipo})$

- **cudaMemcpy**.- copia memoria hacia y desde el device.

Ejemplo:

```
cudaMemcpy(destino, origen, tamaño_de_memoria, indicador_flujo_de_inf)
```

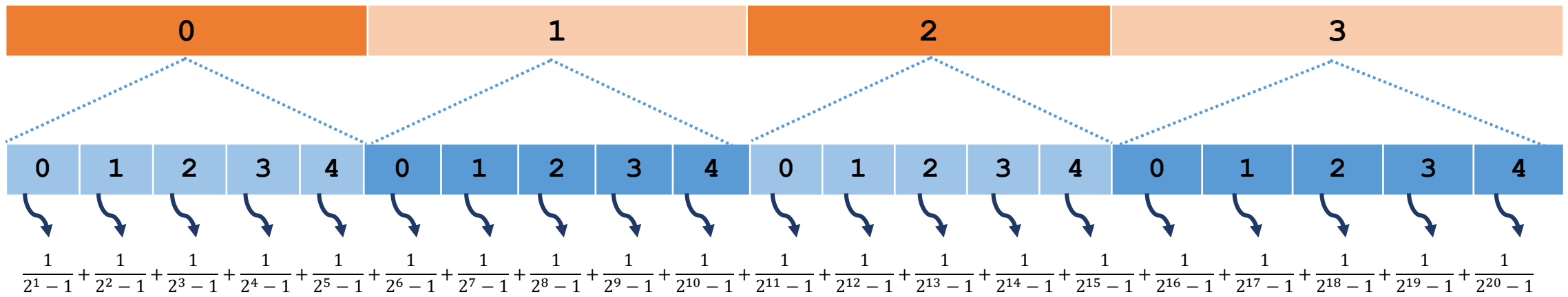
Donde Indicador= cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost,
cudaMemcpyDeviceToDevice

- **cudaFree**.- libera la memoria reservada por un apuntador.

Ejemplo:

```
cudaFree (apuntador)
```

Caso 1. X bloques con numHilos c/u



`tid=(blockIdx.x*blockDim.x)+threadIdx.x`

Caso 1. X bloques con numHilos c/u

```
#define noTerminos 1000000000 // de la formula de Erdos
...
int numHilos = 1024;
...
int numBloques = divEntera(noTerminos, numHilos);
int numCalculos = noTerminos;
...
__global__ void calculoErdos(double* a) {
    int tid = (blockIdx.x*blockDim.x)+ threadIdx.x;
    if (tid < noTerminos) {
        a[tid] = 1.0 / (pow(2,tid+1)-1);
    }
}
```

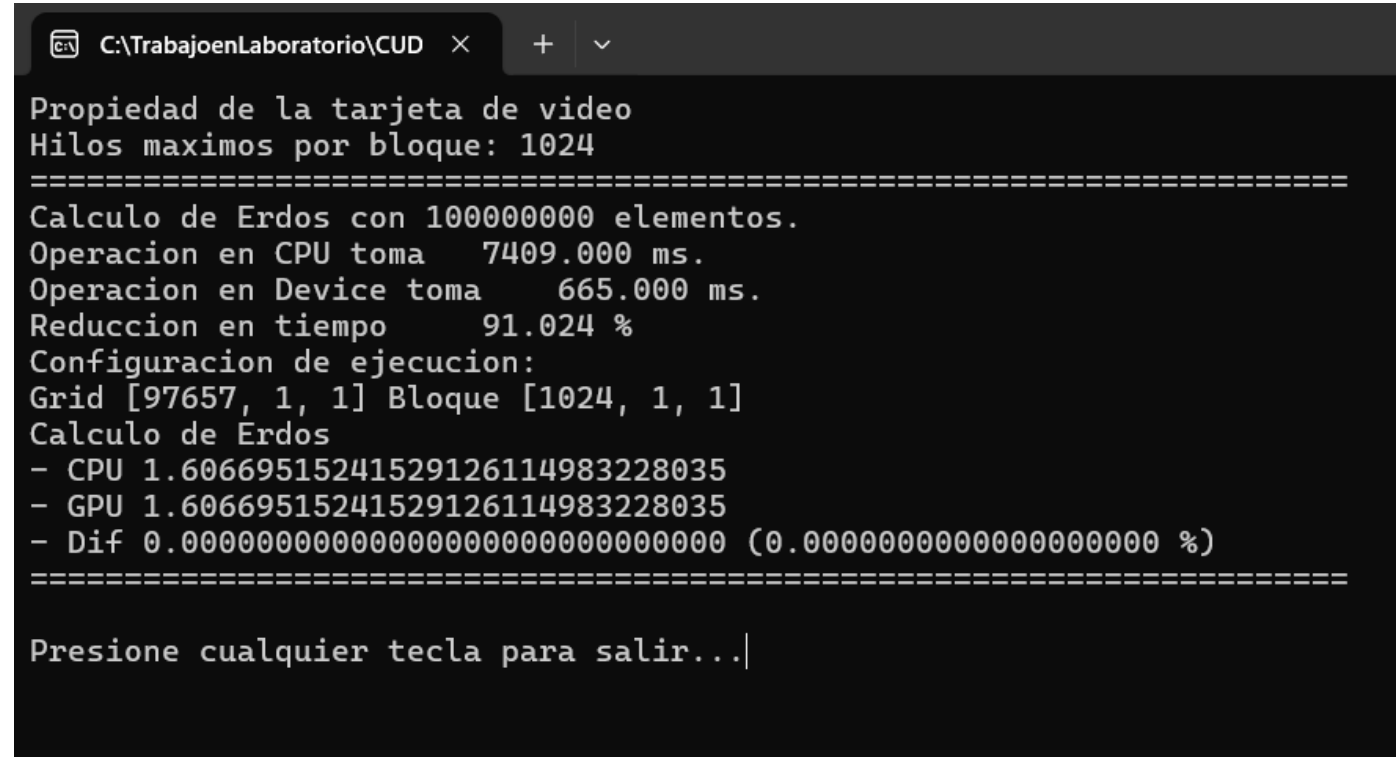
Caso 1. X bloques con numHilos c/u

```
__global__ void sumarErdos(double* a, double* acum, int numElem) {
    int tid = blockIdx.x;
    int inicio = (blockIdx.x * numElem);
    double suma = 0;
    for (int i = 0; i < numElem; i++) {
        suma = suma + a[inicio + i];
    }
    acum[tid] = suma;
}

__global__ void totalErdos(double* acum, double* total, int numElem) {
    double suma = 0;
    for (int i = 0; i < numElem; i++) {
        suma = suma + acum[i];
    }
    *total = suma;
}
```

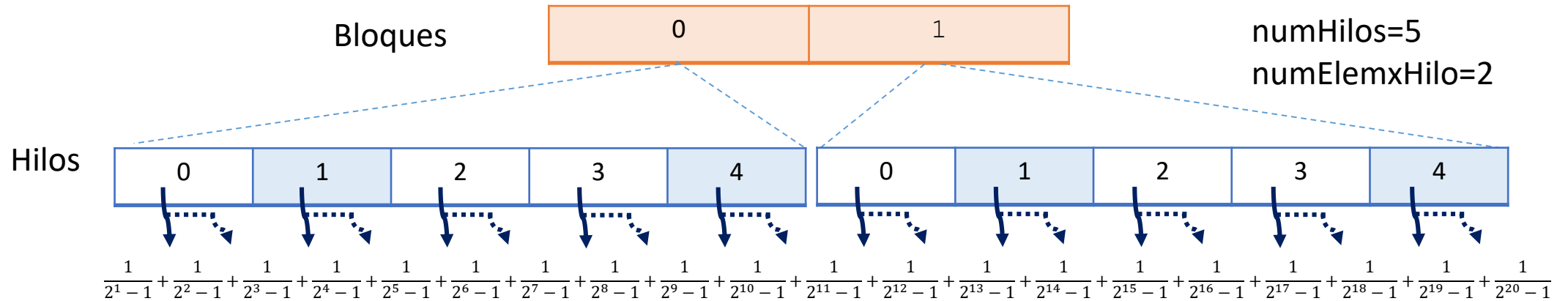
Caso 1. X bloques con numHilos c/u

```
calculoErdos << <dimGrid, dimBlock >> > (dElementos);  
sumarErdos << < dimGrid, 1 >> > (dElementos, dSumaxBloque, numHilos);  
totalErdos << <1, 1 >> > (dSumaxBloque, dTotal, numBloques);
```



```
C:\TrabajoLaboratorio\CUD >  
Propiedad de la tarjeta de video  
Hilos maximos por bloque: 1024  
=====  
Calculo de Erdos con 1000000000 elementos.  
Operacion en CPU toma 7409.000 ms.  
Operacion en Device toma 665.000 ms.  
Reduccion en tiempo 91.024 %  
Configuracion de ejecucion:  
Grid [97657, 1, 1] Bloque [1024, 1, 1]  
Calculo de Erdos  
- CPU 1.60669515241529126114983228035  
- GPU 1.60669515241529126114983228035  
- Dif 0.00000000000000000000000000000000 (0.00000000000000000000000000000000 %)  
=====  
Presione cualquier tecla para salir...|
```

Caso 2. X bloques con numHilos que atienden a numElemxHilo c/u



blockIdx.x	threadIdx.x	tid	Terminos calculados
0	0	0	0, 1
0	1	1	2, 3
0	2	2	4, 5
0	3	3	6, 7
0	4	4	8, 9
1	0	5	10, 11
1	1	6	12, 13
1	2	7	14, 15
1	3	8	16, 17
1	4	9	18, 19

$$\text{tid} = (\text{blockIdx.x} * \text{blockDim.x}) + \text{threadIdx.x}$$

$$\text{PrimerElemento} = \text{tid} * \text{numElemxHilo}$$

Caso 2. X bloques con numHilos que atienden a numElemxHilo c/u

```
#define noTerminos 1000000000 // de la formula de Erdos
#define elemxHilo 5
...
int numHilos = 1024;
...
int numBloques = divEntera(noTerminos, numHilos * elemxHilo);
int numCalculos = numBloques * numHilos;
...
```

Caso 2. X bloques con numHilos que atienden a numElemxHilo c/u

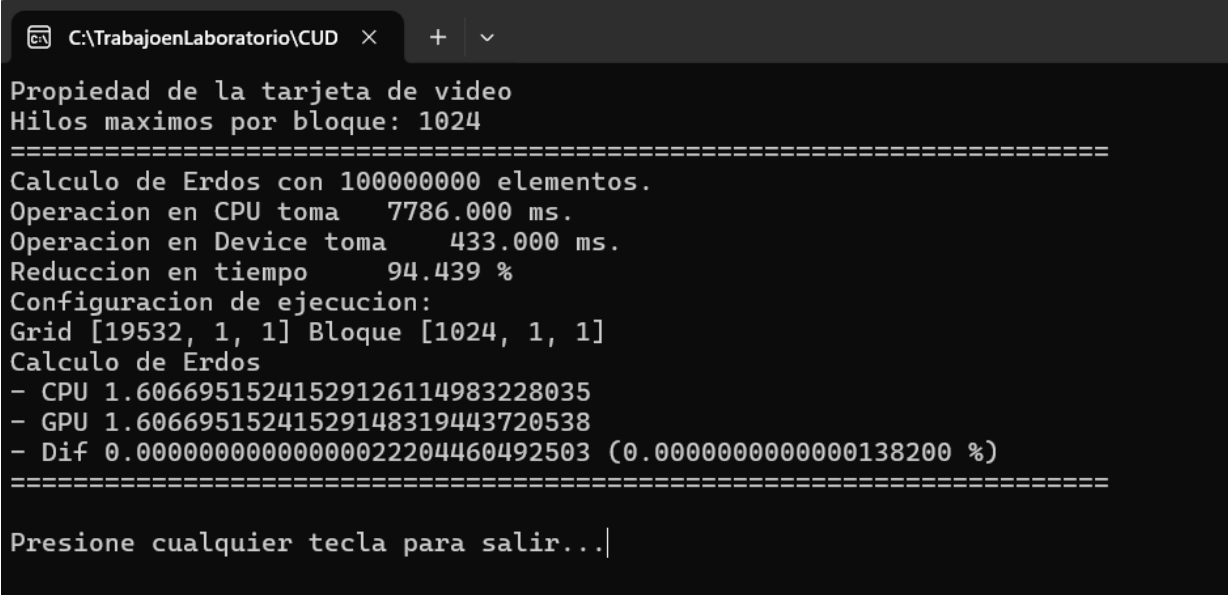
```
__global__ void calculoErdos(double* a) {  
    double valor = 0;  
    double suma = 0;  
    int tid = (blockIdx.x * blockDim.x) + threadIdx.x;  
    int inicio = tid * elemxHilo;  
    if (inicio < noTerminos) {  
        for (int i = 0; i < elemxHilo; i++) {  
            if ((inicio + i) < noTerminos) {  
                valor = inicio + i + 1;  
                suma = suma + (1.0 / (pow(2, valor) - 1));  
            }  
        }  
    }  
    a[tid] = suma;  
}
```

Caso 2. X bloques con numHilos que atienden a numElemxHilo c/u

```
__global__ void sumarErdos(double* a, double* acum, int numElem) {  
    int tid = blockIdx.x;  
    int inicio = (blockIdx.x * numElem);  
    double suma = 0;  
    for (int i = 0; i < numElem; i++) {  
        suma = suma + a[inicio + i];  
    }  
    acum[tid] = suma;  
}  
  
__global__ void totalErdos(double* acum, double* total, int numElem) {  
    double suma = 0;  
    for (int i = 0; i < numElem; i++) {  
        suma = suma + acum[i];  
    }  
    *total = suma;  
}
```


Caso 2. X bloques con numHilos que atienden a numElemxHilo c/u

```
calculoErdos << <dimGrid, dimBlock >> > (dElementos);  
sumarErdos << < dimGrid, 1 >> > (dElementos, dSumaxBloque, numHilos);  
totalErdos << <1, 1 >> > (dSumaxBloque, dTotal, numBloques);
```



```
C:\TrabajoLaboratorio\CUD >  
Propiedad de la tarjeta de video  
Hilos maximos por bloque: 1024  
=====  
Calculo de Erdos con 100000000 elementos.  
Operacion en CPU toma 7786.000 ms.  
Operacion en Device toma 433.000 ms.  
Reduccion en tiempo 94.439 %  
Configuracion de ejecucion:  
Grid [19532, 1, 1] Bloque [1024, 1, 1]  
Calculo de Erdos  
- CPU 1.60669515241529126114983228035  
- GPU 1.60669515241529148319443720538  
- Dif 0.00000000000000022204460492503 (0.00000000000000138200 %)  
=====  
Presione cualquier tecla para salir...|
```

Bibliografía

- Documentación **CUDA C++ Programming Guide** NVIDIA. 2024
<https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html>
- Sitio **CUDA Toolkit Documentation** NVIDIA, 2024.
<https://docs.nvidia.com/cuda/index.html>
- Storti, Duane; Yurtoglu, Mete. **CUDA for Engineers:An Introduction to High-Performance Parallel Computing**. Addison Wesley. 2015.
- Cheng, John; Grossman, Max; McKercher. **Professional CUDA C Programming**. Edit. Wrox. 2014.
- Sanders, Jason; Kandrot, Edward. **CUDA by Example:An Introduction to General-Purpose GPU Programming**. Addison Wesley. 2011.
- Kirk, David; Hwu, Wen-mei. **Programming Massively Parallel Processors: A Hands-on Approach**. Elsevier. 2010.

Gracias por su atención



**U.A.Q. Fac. de Informática
Campus Juriquilla**

**Dra. Sandra Luz Canchola Magdaleno
sandra.canchola@uaq.mx
Cel. 442-1369270**

**Dra. Reyna Moreno Beltrán
reyna.moreno@uaq.mx**