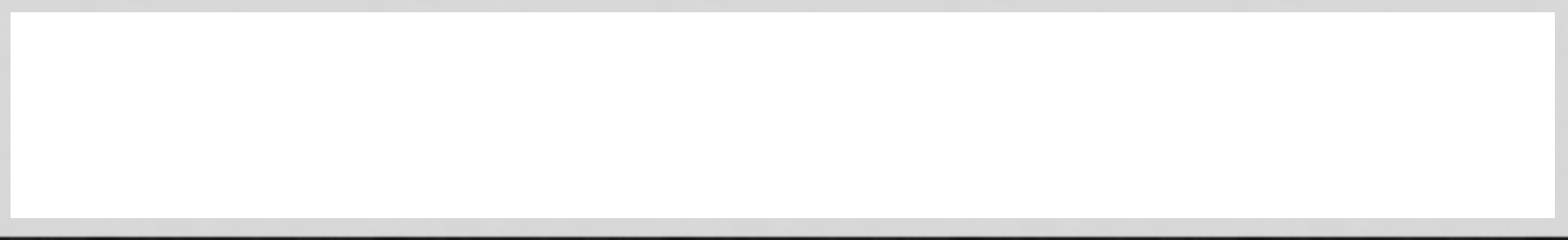
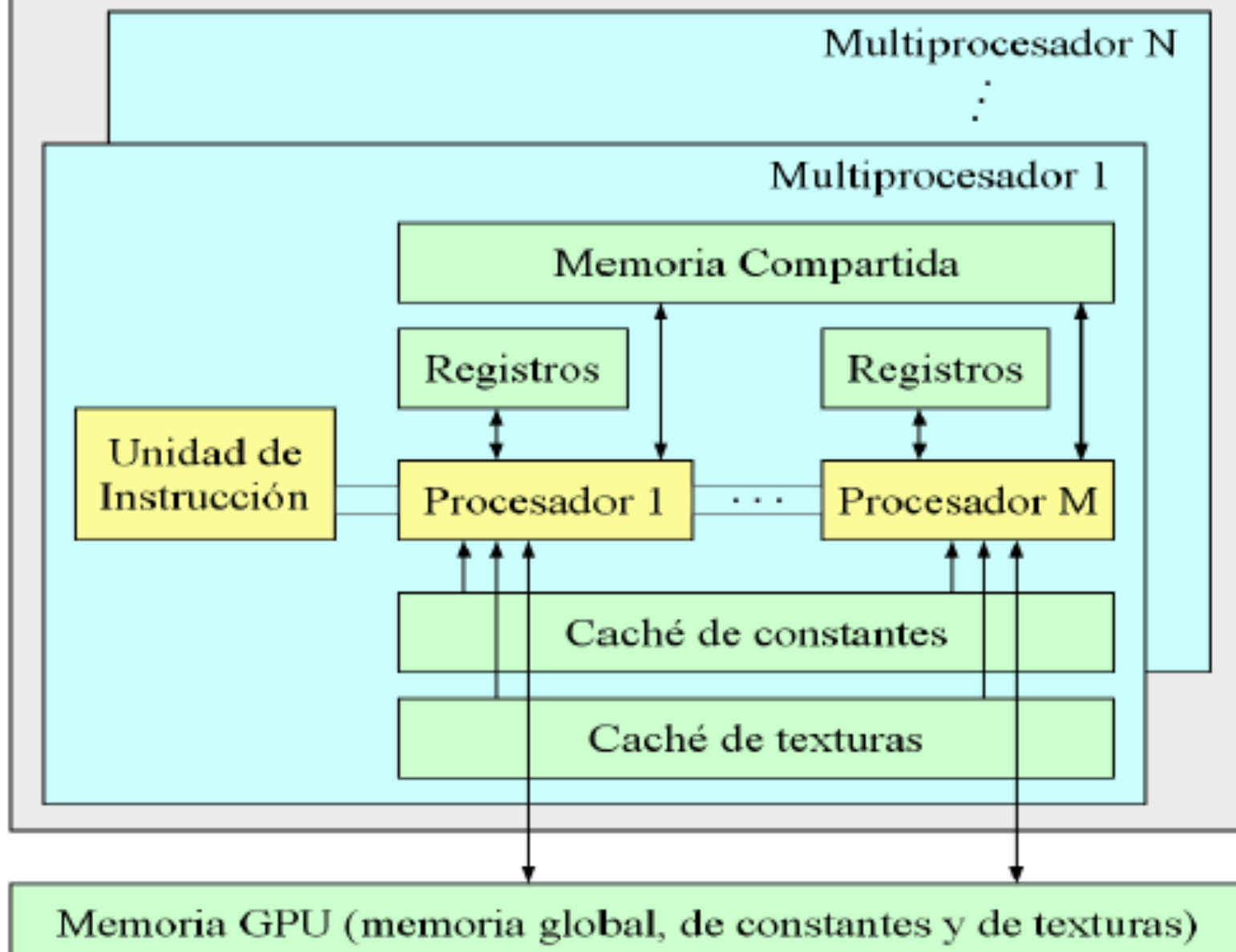


SHARED MEMORY



- 
- Los programas que usan memoria compartida se realiza una copia del contenido de la memoria global a la memoria compartida y viseversa.
 - ¿De qué tamaño es la memoria compartida?
 - ¿Qué permisos cuenta la memoria compartida?
 - ¿Cuál es el alcance de la memoria compartida?

GPU



DECLARACIÓN DE LA MEMORIA COMPARTIDA

- La memoria compartida se declara en el kernel usando el tipo de variable

`__shared__`

- La memoria compartida es memoria por bloque
- Cada hilo únicamente accede a un elemento del arreglo a la vez.

```
__shared__ int nomarreglo[BLOCKSIZE];
```

ACCESO A LA MEMORIA COMPARTIDA

1. Transferir el arreglo de la memoria global a la memoria compartida.
2. Transferir el contenido del arreglo de la memoria compartida a la memoria global.

Nota: el acceso a la memoria compartida es significativamente más rápido que el acceso a la memoria global.

Capacidad: La memoria compartida (16K-48K por MP).

BARRERA DE SINCRONIZACIÓN DE THREAD

- Una barrera de sincronización de hilos se coloca al final del kernel

```
void __syncthreads()
```

Función:

Sincronizar todos los hilos de un mismo bloque

- Establece barrera de sincronización entre todos los hilos del bloque
- Se usa para evitar inconsistencias en el acceso a memoria compartida

```
//copy global memory to shared memory  
array_MShared[tx]=array_MGlobal[idx];  
array_MShared[tx]=array_MGlobal[tx]*10;
```

```
//copy shared memory back to global memory  
array_MGlobalOut[idx]=array_MShared[tx];  
__syncthreads();
```

DOT

```
__global__ void kernel(int *d, int n)
{
    __shared__ int s[64];
    int t = threadIdx.x;
    int tr = n-t-1;
    s[t] = d[t];
    __syncthreads();
    d[t] = s[tr];
}
```



```
int main(void)
{
    const int n = 64;
    int a[n], r[n], d[n];

    for (int i = 0; i < n; i++) {
        a[i] = i;
        r[i] = n-i-1;
        d[i] = 0;
    }

    int *d_d;
    cudaMalloc(&d_d, n * sizeof(int));

    cudaMemcpy(d_d, a, n*sizeof(int), cudaMemcpyHostToDevice);
    kernel<<<1,n>>>(d_d, n);
    cudaMemcpy(d, d_d, n*sizeof(int), cudaMemcpyDeviceToHost);
}
```

```
for (int i = 0; i < n; i++)  
    if (d[i] != r[i])  
        printf("Verificar- Hay un error");  
  
}
```

SUMA DE VECTORES

¿Cómo sería la suma de vectores utilizando la memoria compartida?

PRACTICA 3:

Realizar la suma de dos vectores de 256×256 elementos.

¿Cómo se definirían los bloques e hilos de la suma de dos vectores de 2048×2048 elementos, utilizando memoria compartida?

Realizar una comparativa de rendimiento (speedup) entre memoria compartida y memoria global

MULTIPLICACIÓN DE MATRICES CON MEMORIA COMPARTIDA

- Realizar la multiplicación y entregar el día 7 de Noviembre una comparativa entre memoria global y memoria compartida. Obtener el speedup