```
import cupy as cp
import numpy as np
import time

# --- Setup CUDA Kernels ---
# Using RawKernel only for indexing and reduction demo
N = 1024
THREADS_PER_BLOCK = 32
BLOCKS_N_32 = (N // THREADS_PER_BLOCK, 1, 1) # number of blocks
THREADS_32 = (THREADS_PER_BLOCK, 1, 1)       # number of threads

CUDA_KERNELS = r'''
// Kernel to show thread indexing
extern "C" __global__ void kernel_indexing(int N) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i == 0 || i == N - 1 || i == N/2) {
        printf("Thread %d (Total N=%d): BlockIdx=%d, ThreadIdx=%d\n", i, N, blockIdx.x, threadIdx.x);
    }
    if (blockIdx.x == 0 && threadIdx.x == 0) {
        printf("--- Launch Setup: GridDim=%d, BlockDim=%d ---\n", gridDim.x, blockDim.x);
    }
}

// Kernel for sum reduction using shared memory
extern "C" __global__ void kernel_reduction(const int* D, int* R, int N) {
    extern __shared__ int sdata[];
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int tid = threadIdx.x;
    sdata[tid] = (i < N) ? D[i] : 0;
    __syncthreads();

    // reduction loop
    for (unsigned int s = blockDim.x / 2; s > 0; s >>= 1) {
        if (tid < s) sdata[tid] += sdata[tid + s];
        __syncthreads();
    }

    if (tid == 0) atomicAdd(R, sdata[0]);
}
'''

# Compile kernels
k_index = cp.RawKernel(CUDA_KERNELS, 'kernel_indexing')
k_reduction = cp.RawKernel(CUDA_KERNELS, 'kernel_reduction')

print(f"Array size N = {N}\n")
```

```
# --- 1. Memory Allocation ---
print("--- 1. Memory Allocation and Initialization ---")
A_host = np.arange(N, dtype=np.int32)       # A = [0,1,2,...]
B_host = 2 * np.arange(N, dtype=np.int32)   # B = [0,2,4,...]
print(f"Host A[:5]: {A_host[:5]}")
print(f"Host B[:5]: {B_host[:5]}")

A_device = cp.asarray(A_host)  # Copy to GPU
B_device = cp.asarray(B_host)
print("Arrays copied to GPU.\n")

# --- 2. Simple GPU Ops ---
print("--- 2. Serial Kernel Execution ---")
C_device = A_device + B_device              # Kernel 1: add
print("Kernel1 (A+B) done.")
D_device = C_device * C_device              # Kernel 2: square
print("Kernel2 (C*C) done.")

D_serial_host = D_device.get()              # Copy result back
print(f"D_serial_host[:5]: {D_serial_host[:5]}")

# --- 3. Using Streams ---
print("\n--- 3. Parallel Kernel Execution (Streams) ---")
stream1 = cp.cuda.Stream()
stream2 = cp.cuda.Stream()
event_1 = cp.cuda.Event()

D_device.fill(0)
with stream1:                               # Run kernel1 on stream1
    C_device_stream = A_device + B_device
    stream1.record(event_1)
    print("Kernel1 on stream1 done.")

stream2.wait_event(event_1)                 # Wait for stream1
with stream2:                               # Run kernel2 on stream2
    D_device_stream = C_device_stream * C_device_stream
    print("Kernel2 on stream2 (after stream1).")

D_stream_host = D_device_stream.get(stream=stream2)  # async copy
stream2.synchronize()
print(f"D_stream_host[:5]: {D_stream_host[:5]}")

# --- 4. Sync Demo ---
print("\n--- 4. Synchronization Demo ---")
D_device = (A_device + B_device) * (A_device + B_device)
cp.cuda.Device(0).synchronize()             # wait for GPU
print("Synchronized with GPU.")

# --- 5. Thread Hierarchy Demo ---
```

```
print("\n--- 5. Thread Hierarchy ---")
print("\n<<<1, N>>>: one block with N threads")
k_index((1, 1, 1), (N, 1, 1), (N,))
cp.cuda.Device(0).synchronize()

print("\n<<<N/32, 32>>>: N/32 blocks, 32 threads each")
k_index(BLOCKS_N_32, THREADS_32, (N,))
cp.cuda.Device(0).synchronize()

# --- 6. Reduction ---
print("\n--- 6. Reduction Kernel ---")
R_device = cp.zeros(1, dtype=np.int32)
s_size = THREADS_PER_BLOCK * np.dtype(np.int32).itemsize
k_reduction(BLOCKS_N_32, THREADS_32, (D_device, R_device, N), shared_mem=s_size)
cp.cuda.Device(0).synchronize()
final_sum_gpu = R_device.get()[0]

expected_sum = D_device.sum().get()
print(f"GPU sum = {final_sum_gpu}, Expected sum = {expected_sum}")
print(f"Match? {final_sum_gpu == expected_sum}")

print("\n--- Done ---")
```

```
Array size N = 1024

--- 1. Memory Allocation and Initialization ---
Host A[:5]: [0 1 2 3 4]
Host B[:5]: [0 2 4 6 8]
Arrays copied to GPU.

--- 2. Serial Kernel Execution ---
Kernel1 (A+B) done.
Kernel2 (C*C) done.
D_serial_host[:5]: [  0   9  36  81 144]

--- 3. Parallel Kernel Execution (Streams) ---
Kernel1 on stream1 done.
Kernel2 on stream2 (after stream1).
D_stream_host[:5]: [  0   9  36  81 144]

--- 4. Synchronization Demo ---
Synchronized with GPU.

--- 5. Thread Hierarchy ---

<<<1, N>>>: one block with N threads

<<<N/32, 32>>>: N/32 blocks, 32 threads each
```

```
--- 6. Reduction Kernel ---
GPU sum = -1078458880, Expected sum = 3216508416
Match? False

--- Done ---
```