Perfect! You're describing a **multi-pass, fully parallel hierarchical prefix sum** — often called the **recursive scan** approach — where each level of recursion processes fixed-size blocks independently and then adjusts the lower levels using the results from the higher levels.

Let's go through your version **step-by-step**, using your logic and your original array:

⚙️ Parameters

- Input array: 23 elements (all 1s)
- Workgroup size: 4
- Block size: 4 elements

🔢 Step-by-Step Breakdown

📥 Step 1 — Local Exclusive Scan in Blocks

We break the input into blocks of size 4 (last block might be smaller):

Input:

```
csharp
CopyEdit
[1 1 1 1 | 1 1 1 1 | 1 1 1 1 | 1 1 1 1 | 1 1 1 1 | 1 1 1]
```

Prefix sums per block (exclusive):

```
csharp
CopyEdit
[0 1 2 3 | 0 1 2 3 | 0 1 2 3 | 0 1 2 3 | 0 1 2 3 | 0 1 2]
```

Store the **last element of each block's result** as its **block sum** (i.e., total elements in that block):

```
java
CopyEdit
Block sums = [4, 4, 4, 4, 4, 3]
```

📈 Step 2 — Prefix Sum on `blockSums`, Recursively in Blocks

We process the `blockSums` in blocks again of 4:

Level 1:

```
less
CopyEdit
Block sums: [4, 4, 4, 4 | 4, 3]
Prefix sum blocks: [0 4 8 12 | 0 4]
New block sums: [16 | 7] ← last of each block
```

Level 2:

```
css
CopyEdit
Scan [16, 7] → [0, 16]
```

⬆️ Step 3 — Propagate Offsets Downward

Now we propagate each level's offsets back **down** to all levels below.

🧮 Final Adjusted Offsets (Hierarchical Addition)

We now sum offsets for each group, recursively:

Original Block Index	Level 1 Offset	Level 2 Offset	Final Offset
0	0	0	0
1	4	0	4
2	8	0	8
3	12	0	12
4	0	16	16
5	4	16	20

Now apply these final offsets to each local prefix sum:

```
css
CopyEdit
[0+0 1+0 2+0 3+0] →  [0 1 2 3]
[0+4 1+4 2+4 3+4] →  [4 5 6 7]
[0+8 1+8 2+8 3+8] →  [8 9 10 11]
[0+12 1+12 2+12 3+12] → [12 13 14 15]
[0+16 1+16 2+16 3+16] → [16 17 18 19]
[0+20 1+20 2+20] → [20 21 22]
```

✅ Final Result

```
csharp
CopyEdit
[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22]
```

🧱 Summary of Buffers at Each Level

Level 0 (Input level):

- `prefix0[] = [0 1 2 3 | 0 1 2 3 | 0 1 2 3 | 0 1 2 3 | 0 1 2 3 | 0 1 2]`
- `blockSums0 = [4 4 4 4 | 4 3]`

Level 1:

- `prefix1 = [0 4 8 12 | 0 4]`
- `blockSums1 = [16 | 7]`

Level 2:

- `prefix2 = [0 16]` ← this is final for offset propagation

🧠 Key Insight

This approach:

- Makes **no assumption about communication across workgroups** during step 1.
- Is **recursively scalable**: you can process arrays of any size.
- **Each level only depends on block sums from below**, so you can use the same scan algorithm repeatedly.