BLUR IMAGE WITH INTEGRAL IMAGE

SEQUENTIAL SOLUTION

```
void blurImageRGB(unsigned char *in_image, unsigned char *out_image, int
 width, int height, int channel, int kernel) {
  int r = (kernel - 1) / 2;

  int integral_img_size = (int)width * height;
  int total_elements = integral_img_size * channel;
  long *integral_img = (long *)calloc(total_elements, sizeof(long));

  // Step 1: Build integral images
  for (int c = 0; c < channel; c++) {
    int channel_offset = integral_img_size * c;

    for (int y = 0; y < height; y++) {
      long rowSum = 0;
      int row_offset = channel_offset + (int)y * width;

      for (int x = 0; x < width; x++) {
        int idx_in = (y * width + x) * channel + c;
        rowSum += in_image[idx_in];

        int idx_ii = row_offset + x;

        if (y == 0) {
          integral_img[idx_ii] = rowSum;
        } else {
          int idx_ii_prev_row = idx_ii - width;
          integral_img[idx_ii] = integral_img[idx_ii_prev_row] + rowSum;
        }
      }
    }
  }
```

```
  // Step 2: Compute blurred image
  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      int x1 = MAX(0, x - r);
      int y1 = MAX(0, y - r);
      int x2 = MIN(width - 1, x + r);
      int y2 = MIN(height - 1, y + r);
      int area = (x2 - x1 + 1) * (y2 - y1 + 1);

      int idx_out_base = ((int)y * width + x) * channel;

      for (int c = 0; c < channel; c++) {
        int channel_offset = integral_img_size * c;
        long sum = integral_img[channel_offset + (int)y2 * width + x2];
        if (x1 > 0) sum -= integral_img[channel_offset + (int)y2 * width +
 (x1 - 1)];

        if (y1 > 0) sum -= integral_img[channel_offset + (int)(y1 - 1) *
width + x2];

        if (x1 > 0 && y1 > 0) sum += integral_img[channel_offset + (int)(
y1 - 1) * width + (x1 - 1)];

        out_image[idx_out_base + c] = (unsigned char)(sum / area);
      }
    }
  }
  free(integral_img);
}
```

SEQUENTIAL SOLUTION

- **Calculate integral image**
- **Blur image using integral image**

98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

Image

98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

Integral Image

SEQUENTIAL SOLUTION

- **Calculate integral image**
  - **O(W x H x C)**
- **Blur image using integral image**
  - **O(W x H x C)**
- **Total time complexity**
  - **O(W x H x C)**

PARALLEL SOLUTION

```cpp
__global__ void horizontal_integral(unsigned char *input, long long *output,
int width, int height, int C, int channel) {
  int row = blockIdx.x * blockDim.x + threadIdx.x;

  if (row >= height) return;

  // Each thread processes one complete row sequentially
  long long sum = 0;
  for (int col = 0; col < width; col++) {
    int idx = (row * width + col) * C + channel;
    sum += input[idx];
    output[row * width + col] = sum;
  }
}
```

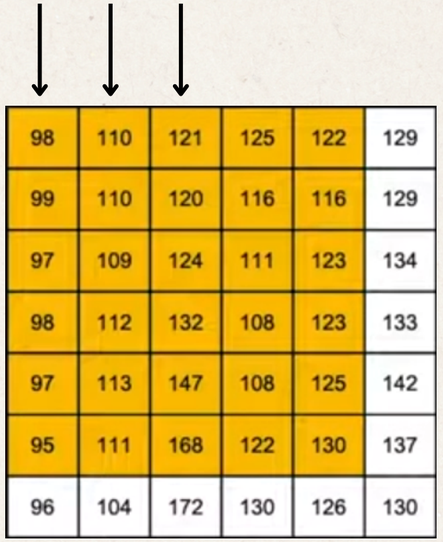
- **Work**
  - **O(W x H)**
- **Span**
  - **O(W)**

PARALLEL SOLUTION

```cpp
__global__ void vertical_integral(long long *input, long long *output, int width, int height) {
  int col = blockIdx.x * blockDim.x + threadIdx.x;

  if (col >= width) return;

  // Each thread processes one complete column sequentially
  long long sum = 0;
  for (int row = 0; row < height; row++) {
    sum += input[row * width + col];
    output[row * width + col] = sum;
  }
}
```



- **Work**
  - **O(W x H)**
- **Span**
  - **O(H)**

PARALLEL SOLUTION

```cpp
__global__ void blur_from_integral(long long **S_device, unsigned char *output
, int width, int height, int channel, int r) {
  int x = blockIdx.x * blockDim.x + threadIdx.x;
  int y = blockIdx.y * blockDim.y + threadIdx.y;

  if (x >= width || y >= height) return;

  int x1 = max(0, x - r);
  int y1 = max(0, y - r);
  int x2 = min(width - 1, x + r);
  int y2 = min(height - 1, y + r);

  int area = (x2 - x1 + 1) * (y2 - y1 + 1);

  for (int c = 0; c < channel; c++) {
    long long *S = S_device[c];

    long long sum = S[y2 * width + x2];
    if (x1 > 0) sum -= S[y2 * width + (x1 - 1)];
    if (y1 > 0) sum -= S[(y1 - 1) * width + x2];
    if (x1 > 0 && y1 > 0) sum += S[(y1 - 1) * width + (x1 - 1)];

    int idx = (y * width + x) * channel + c;
    output[idx] = (unsigned char)(sum / area);
  }
}
```

- **Work**
  - **O(W x H x C)**
- **Span**
  - **O(C)**

PARALLEL SOLUTION

- **Total work**
  - **O(W x H x C)**
  - **Work Efficient !**
- **Total span**
  - **O(C x (W + H))**
- **Sequential span**
  - **O(W x H x C)**
- **Parallel span**
  - **O(C x (W + H))**
- **Degree of parallelism**
  - **O(W x H x C) / O(C x (W + H))**

    = O(W x H) / O(W + H)

COMPARE

TEST BLUR IMAGE

- **Image size 4000 x 3000**
- **Parallel faster than sequential ~2.7x**

SEQUENTIAL SOLUTION

```
Image loaded: 4000x3000 (3 channels)
Blurring done in 0.4880 seconds (kernel=30)
```

PARALLEL SOLUTION

```
Image loaded: 4000x3000 (3 channels)
Blurring done in 0.1760 seconds (kernel=30)
```

THANK YOU!