Distributed Gradient-Domain Processing of Planar and Spherical Images

MICHAEL KAZHDAN
Johns Hopkins University

DINOJ SURENDRAN and HUGUES HOPPE
Microsoft Research

Supplemental Material: Serializability of the distributed solver

To prove that our method provides a correct implementation of a Gauss-Seidel solver, we need to show that though we implement the relaxations in a parallel manner, there exists a serial Gauss-Seidel implementation that achieves the same results.

This requires demonstrating that for each of the $1 \leq i \leq k$ relaxation updates, there is an indexing σ_i of pixels such that whenever the i-th relaxation at pixel p depends on the value at pixel q, then pixel q must have already been updated i times if $\sigma_i(q) < \sigma_i(p)$, and pixel q must have been updated $(i-1)$ times if $\sigma_i(q) \geq \sigma_i(p)$.

We begin with a description of the indexing that would serialize our Gauss-Seidel relaxation if we allowed the synchronizations to occur after each row is updated, and then describe how this indexing can be extended to support our solver which only synchronizes once per window advancement.

Serializability of a simple distributed solver

In the simplest implementation we can sweep through the entire row starting at one end of the band and terminating at the other (moving left-to-right on even rows and right-to-left on odd rows.) Before completing the update of the band, we synchronize the data with the neighbors and then complete the processing of the row. We then proceed to the row two above, and repeat the same process, continuing until we have updated k rows.

Whether we process an even row or an odd row for the k-th time, it is always the case that as we start the processing, pixels in the two rows above will have already been updated exactly k times, and pixels in the current row and the two rows below will have been updated $(k-1)$ times. (This property is guaranteed by temporal blocking).

Furthermore, for even rows, it is generally the case that as we update the pixels in a row for the k-th time, the pixels to the left have been updated k times and the pixels to the right have been updated $(k-1)$ times. This property is only violated near the right boundary where the pixels that we read from the right buffer region will already have been updated k times. (A symmetric situation occurs in the processing of the odd rows.)

Thus, the distributed relaxations can be serialized through an interleaved and boustrophedonic ordering. Assuming P processors, each with bands of width W, this ordering is defined by setting the index of the (x,y)-th pixel in the p-th band to:

$$\sigma(x,y,p) = \begin{cases} W \cdot P \cdot y + (P \cdot x + p) & \text{if y is even} \\ W \cdot P \cdot y + W \cdot P - 1 - (P \cdot x + p) & \text{if y is odd} \end{cases}$$

An example of this indexing for $P = 4$, $W = 5$ is shown in Figure 1.

Serializability of our distributed solver

What makes our solver more complicated is that we only perform one synchonization as we process the k rows. This requires the different updates to start and end their relaxations at different offsets from the band boundaries, with the i-th relaxation beginning its updates $2(k-i)$ pixels before the start of the band and ending its updates $2(k-1)$ pixels before the end. As a result our serialization requires defining a different indexing function σ_i for each of the $1 \leq i \leq k$ relaxation updates.

Note that the use of a lane-based solver does not affect the indexing as it is always the case that when updating a pixel in row $(l+2)$ for the $(i+1)$-st time its neighbors in row l have already been updated i times, even if the remaining pixels in row l have not.

Thus, we serialize our processing by adapting the indexing σ to the i-th iteration, defining the indexing σ_i by offsetting the indexing of pixels in the even rows by $2(k-i)$ indices to the left, and shifting the index of pixels in the odd rows by $2(k-i)$ indices to the right.

As a result, for the last of the k Gauss-Seidel iterations, the indexing will remain unchanged, $\sigma_k = \sigma$. However, for earlier iterations, the indexing will resemble the original indexing σ alternatively shifted to the left and to the right, with the size of the shift increasing for earlier updates. An example for $i = k-1$ is shown in Figure 2.

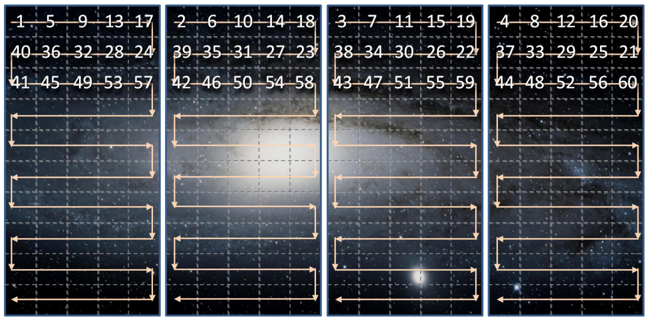

Figure 1: The basic interleaved and boustrophedonic ordering used to serialize our Gauss-Seidel relaxation.

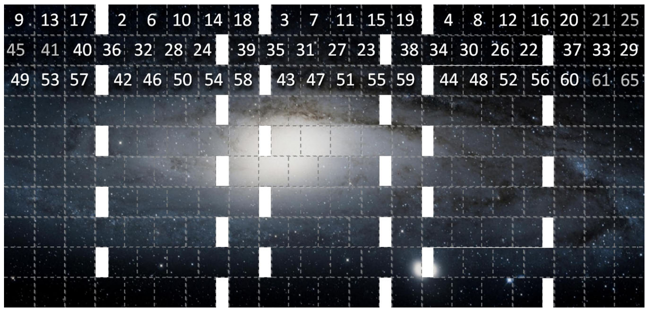

Figure 2: The shifted, interleaved, and boustrophedonic ordering used to serialize iteration $k-1$ in our Gauss-Seidel relaxation.