Supplementary For “A Fast GPU Schedule For À-Trous Wavelet-Based Denoisers”

Reiner Dolp, Johannes Hanika and Carsten Dachsbacher

Karlsruhe Institute of Technology, Germany

Outline In this supplementary material, we discuss additional results for our padding-free bank conflict resolution strategy outlined in section 3.5 of the paper. More specifically, section 1 discusses the bitwidth correction factor $\hat{c}_b$, section 2 discusses how to normalize the order of banks within a subgroup, and section 3 discusses how to reshape bank patterns.

1 Bitwidth Correction

Equation 1 in the paper analyses the relationship of two memory slices—the bitwidth b_s of a row $1 \times W_s$ in shared memory, and the bitwidth of a shared memory slice b_c accessed by the i invocations participating in bank conflict resolution—using their index spaces. Consequently, assuming the indexed elements in each slice have identical bitwidth. However, the bitwidth of the indexed elements may differ for several reasons. First, because each array element in shared memory may cover multiple banks. Second, because a device may resolve bank conflicts inherently on a finer granularity than a subgroup or do so depending on the bitwidth of the memory transaction.

To match the bitwidth of elements in both index spaces, we scale the second index space to

$$\hat{c}_b = \frac{b_c}{b_s/W_s}.$$

Applying this correction factor scales the height n of a bank pattern tile by factor $\hat{c}_b/i$. As a practical consequence, our bank conflict resolution strategy for b-bit memory accesses resolves bank conflicts for any $b' \geq b$-bit memory accesses,ensuring our method is independent of the load-store instructions selected by the shader compiler.

2 Normalization of Bank Pattern Tiles

To this point, we have only noted that each column within a subset contain the identical set of non-conflicting banks, ignoring the position of banks within each column. In this subsection, we derive how to undo the permutation of banks within a column, and the barrel shift unique to each column – knowledge which we will leverage in the next subsection to reshape bank pattern tiles.

By generalizing the comparison of strides in equation (1) of the paper to allow for offsets within banks we can pose a *row query* through

$$n \cdot \hat{c}_b = m \cdot s + T_x \iff n \cdot \hat{c}_b - m \cdot s = -\gcd(s, \hat{c}_b).$$

and by allowing an offset within the index access function, we can pose a *column query* through

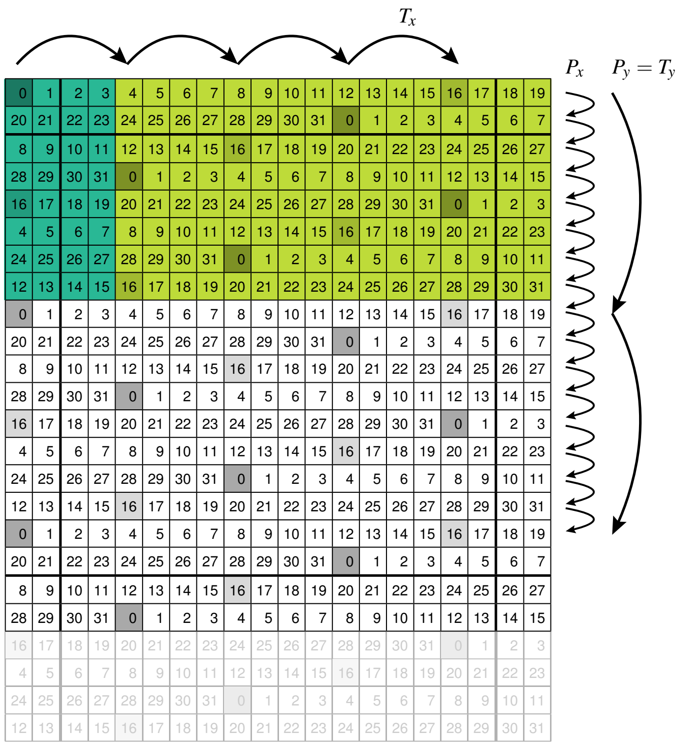
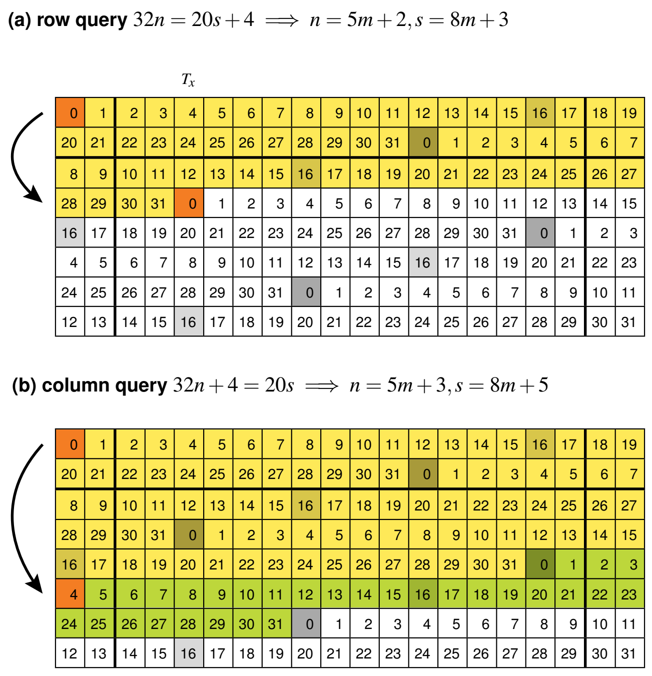


Figure 1: *Larger reproduction of figure 1(d) in the paper showing an assignment of 32 shared memory banks to words within an overlapped tile with halo radius $r = 2$ and workgroup size 16×16. Eight consecutive lines form a bank pattern, which repeats vertically. Within a bank pattern, four rows form a bank pattern tile repeating horizontally with banks barrel shifted by three vertically.*



(a) row query $32n = 20s + 4 \implies n = 5m + 2, s = 8m + 3$
T_x
(b) column query $32n + 4 = 20s \implies n = 5m + 3, s = 8m + 5$

Figure 2: *Geometric interpretation of row and column queries using the shared memory footprint of fig. 1.* **(a)** *The solution of the row query can be read as “Each bank pattern tile consists out of eight rows and cycles five times through the shared memory banks. In column four, the zero-th bank is hit after a shift by three rows, which requires two cycles through the shared memory banks.” Hence, columns within each subset are barrel shifted by three. The i-th bank pattern tile can be made identical to the first bank pattern tile using a barrel shift of $-3i$, equal to an adaptive modular mapping of $i \mapsto i - 3\%8$.* **(b)** *Analogously, the solution to the column query can be interpreted as “The fourth bank in the zero-th column is hit in row five after three cycles through shared memory banks are completed.” The permutation of banks within a column can be inverted using the affine modular mapping $i \mapsto 5i\%8$.*

$$n \cdot \hat{c}_b + T_x = m \cdot s \iff n \cdot \hat{c}_b - m \cdot s = \gcd(s, \hat{c}_b).$$

Thus, information about row positions is queried using negative values on the right-hand side of the equation, and information about columns is queried using positive values. A row query answers the question *“When is bank zero in column T_x?”*, while a column query answers the question *“When is bank T_x in column zero?”*. The result of a row query can be used to undo the barrel shift unique to each column within a subset, making columns within a subset identical. A column query can be used to find the modular stride within a column and can, therefore, be used to undo the permutation within a column, which will sort banks within a column in ascending order from top to bottom.

Our choice of the greatest common divisor as an offset in row queries is motivated by the fact that it equals the offset of the next column within the subset. In contrast, we choose the greatest common divisor for column queries, since it is equal to the index of the second smallest bank index within the subset. Figure 2 gives a geometric intuition into the equations of row and column queries and their solution.

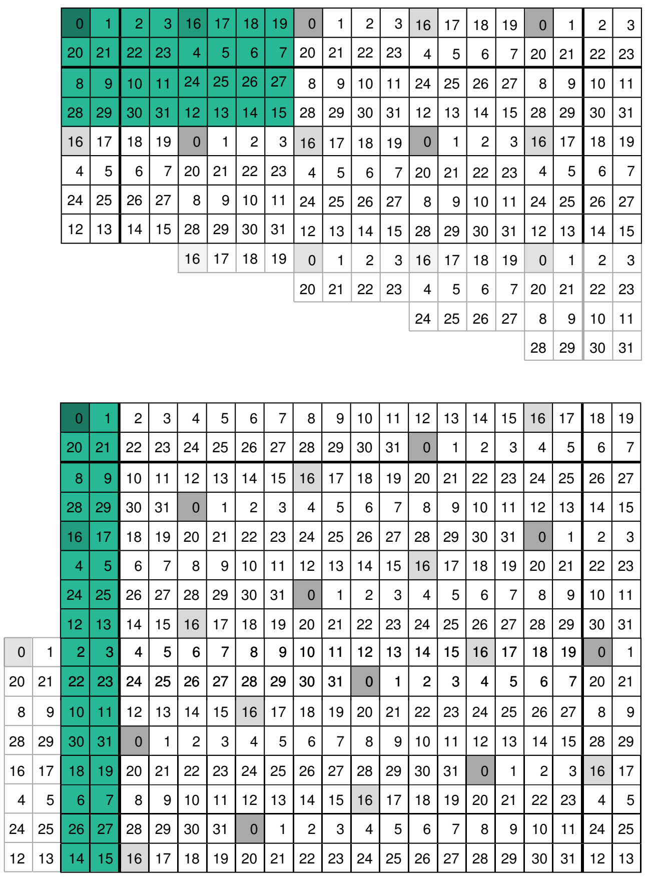


Figure 3: *Reshaping of the bank pattern tile with shape 4×8 in fig. 1.* **(a)** *reshaping to 8×4 through vertical adaptive modular mapping with a granularity of bank pattern tiles.* **(b)** *reshaping to 2×16 through horizontal adaptive modular mapping with a granularity of bank patterns, respectively bank pattern tiles.*

3 Reshaping Bank Pattern Tiles

To reshape bank pattern tiles, we apply a multi-level reshaping strategy. We first apply an adaptive modular mapping on the granularity of bank patterns tiles and then select columns from neighboring bank pattern tiles.

A bank pattern tile of any shape can be reshaped to a single 1×32 column by barrel shifting the i-th bank pattern by i columns to the left. Similarly, a bank pattern tile of width $N \times M$ can be reshaped to $(N/2) \times 2M$ by a barrel shift of $iM/2$. By applying a barrel shift of $iN/2$ to the i-th bank pattern tile, a bank pattern tile can be reshaped to $2N \times (M/2)$. Figure 3 shows an example for bank pattern reshaping along both axis.