Chapter 5

Register Packing

In this chapter we introduce a novel AVX-512 in-register file storage layout that enables single instruction broadcasting, regardless of the vector lane the value is stored in. Both the packing scheme and the method for broadcasting are detailed in Section 5.1. The strategy is evaluated against the reference LIBXSMM on the PyFR suite and synthetic suite. Finally, Intel VTune profiling results are used to further discuss the results.

5.1 Solution

5.1.1 AVX-512 Shuffle Instruction

The storage layout was designed to be able to use the *VSHUFF64X2* AVX-512 instruction. On Skylake-SP, it has a 3 cycle latency [11]. The instruction has two source vector register operands, an 8-bit integer operand and one destination vector operand. The 8-bit integer is encoded at compilation time.
The vector registers are split into sections of 128-bits, so each source and destination register has four sections. The *VSHUFF64X2* instruction copies 128-bit sections from the source registers to the destination.
The 8-bit integer, called the *selector*, selects which source sections to copy for a destination register. The selector is split into four groups of 2-bits. The lowest 2-bits select for the lowest 128-bits in the destination, and the highest 2-bits select for the highest 128-bits in the destination.
The lowest 4-bits of the selector, select for the first two 128-bit sections in the destination register, which are chosen from the first source register. A 2-bit selector can choose any of the four 128-bit sections from the source register.
So the lowest half (256-bits) of the destination register is given two 128-bit sections from the *first* source register. The 128-bit sections chosen can be the same or different. Similarly, the highest 4-bits of the selector choose 128-bit sections from the second source register. The highest half (256-bits) of the destination register is given two 128-bit sections from the *second* source register. The entire operation is illustrated in Figure 5.1 [1].

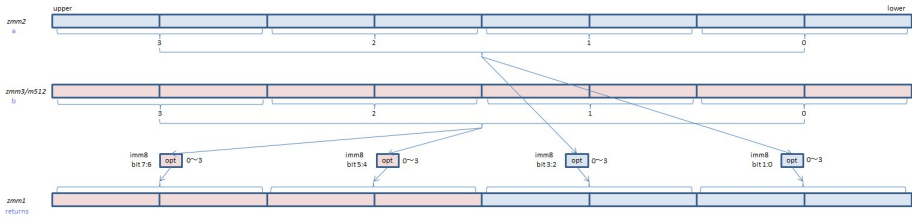
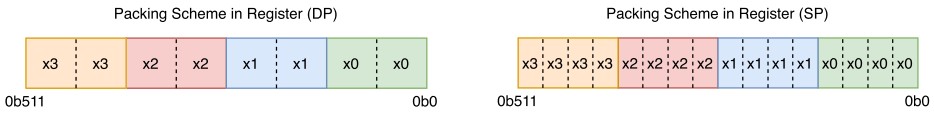


Figure 5.1: Illustration of VSHUFF64X2 [1]

5.1.2 Single Instruction Broadcasting

To make use of the *VSHUFF64X2* instruction for a single-instruction broadcast, the 128-bit sections have to contain *only* the desired value to be broadcast. This means that the value must be repeated for 128-bits. For double precision, this means storing the value twice. For single precision, the value must be stored four times. The logical register layout is shown in Figure 5.2, which shows that the are four 128-bits sections in a register. A trade-off is made by repeating values to obtain a single-instruction broadcast, that uses only one temporary register to hold the broadcasted value. Alternative methods could pack more value in the registers, but would require more instructions and temporary registers to broadcast the values.



(a) Double Precision (b) Single Precision

Figure 5.2: Layout within Register for DP and SP values

Figure 5.3 shows how the *VSHUFF64X2* instruction can be used with the logical storage layout from Figure 5.2 to achieve a single-instruction broadcast, regardless of which lanes (groups of sequential lanes due to repetition) the value is stored in. The same register is used as **both** source operands, which can be seen in Figure 5.3 by both the source registers being *zmm*0. This allows the selector to choose the same 128-bit source section, for all four 128-bit sections in the destination register. The end result is that the selected source section is broadcast to all four sections - a $1-4$ broadcast operation. The same process would apply to SP, and would also be a $1-4$ broadcast, as the SP value must be repeated four times to fill the 128-bit section.

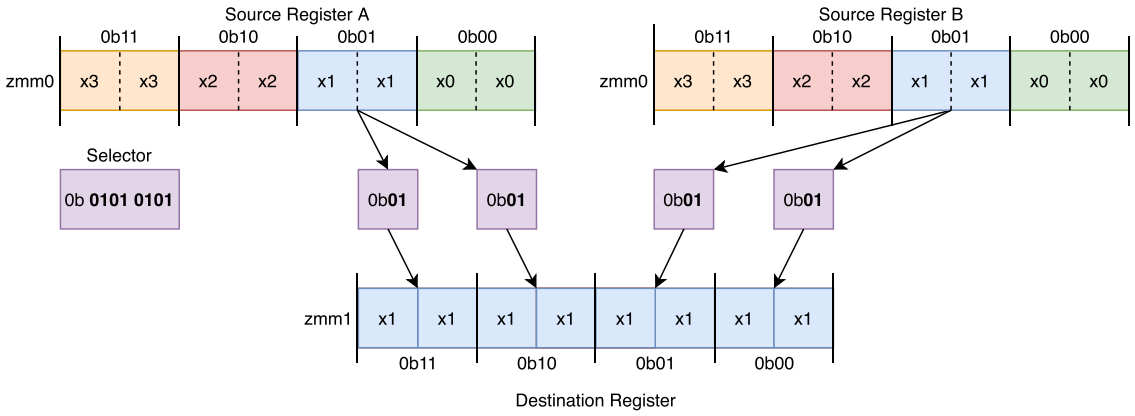


Figure 5.3: Using VSHUFF64X2 to Broadcast

Figure 5.4 showcases the four possible broadcasts and the corresponding selector values required to achieve them. The binary values show that a 2-bit value is repeated, which is due to the same source 128-bit section being chosen for all four destination sections.

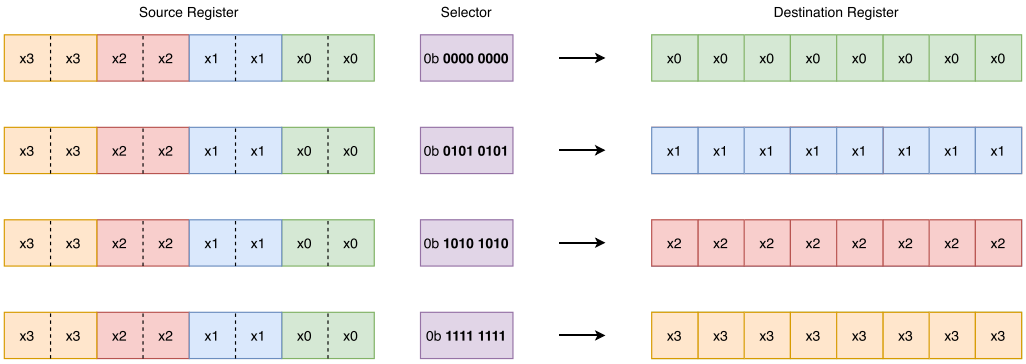


Figure 5.4: The four possible effective broadcasts

5.1.3 Register Packing for the MM routine

By fully storing **A** in the register file with the packing scheme, no memory access is required for **A** during the MM routine. The adaptation made to the routine from Section 2.3.2 is to broadcast the required value of **A** from the *register file* using the *VSHUFF64X2* instruction. Figure 5.5 shows that the 31^{st} register is used as the temporary register to hold the broadcasted value. This is then multiplied with the stride of **B** (loaded from memory) in the FMA to calculate (part of the accumulation of) the stride of **C**.

Hypothesis

1: Kernels for operator matrices where the number of unique non-zeros (U), is bound by $U \leq 31$, should not have decreased performance when using register packing compared to other strategies deployed by LIBXSMM. The run-time broadcasting should *not* add to the critical path of execution.
2: Kernels for operator matrices where the number of unique non-zeros (U), is bound by $31 < U \leq 120$, should have increased performance compared to other strategies deployed by LIBXSMM.

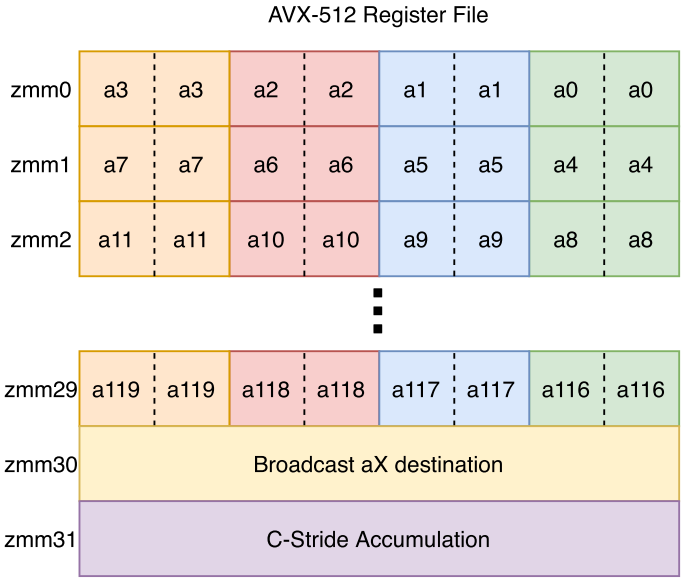


Figure 5.5: The Vector Register File logical layout when using register packing for the SpMM routine