Extended Example of CDPEAC Algorithm

This document gives an extended example of the operation of the computation stage of the vectorised Laplace algorithm described in Section 4 of the Technical Report "Experiences with CDPEAC: Vectorising a Laplace Iteration". The example shows diagrammatically how data is moved between VUs over the course of the computation of the first column of each subgrid. We assume that the communications portion of the iteration has already been performed.

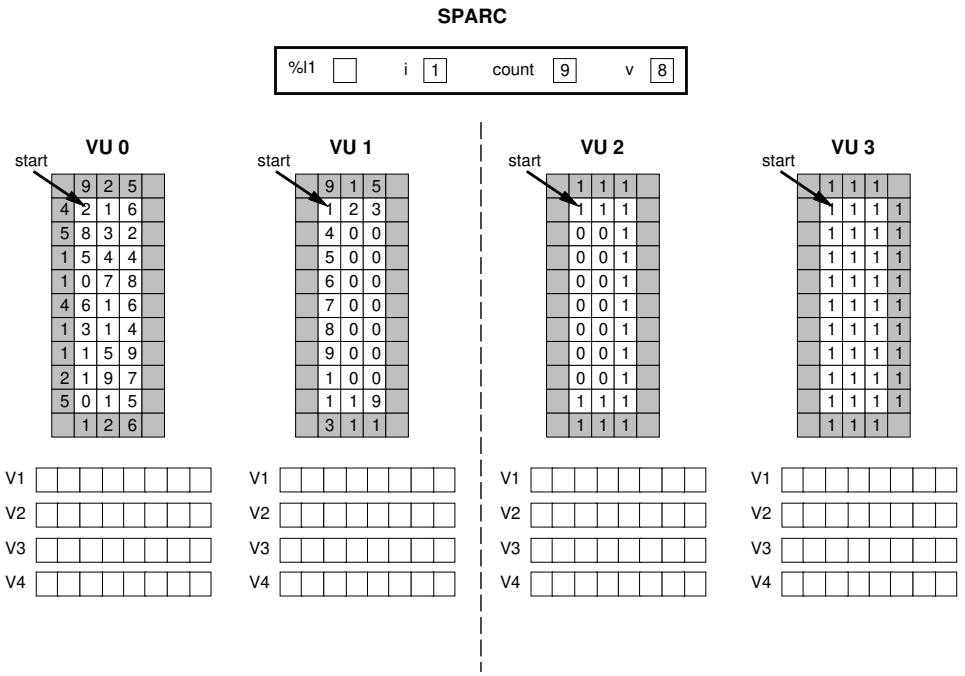

Figure 14: We initialise `i` to 1, `count` to the number of unprocessed elements in column 1, `v` to the vector length. `start` points at the first element to be processed

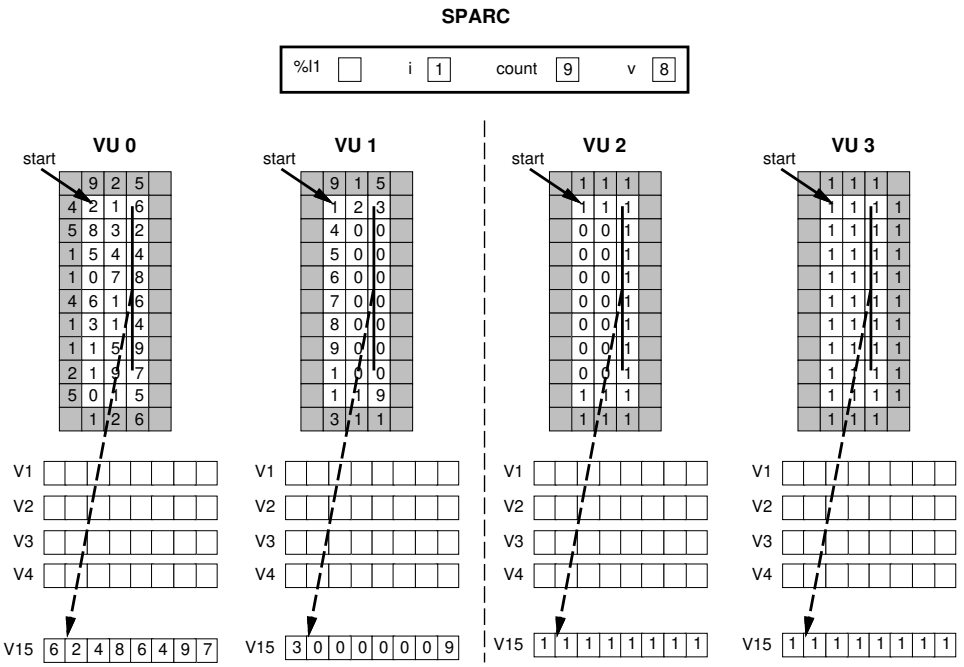

Figure 15: As the first stage of the process of loading left neighbours into V1 we load rightmost columns into the temporary V15

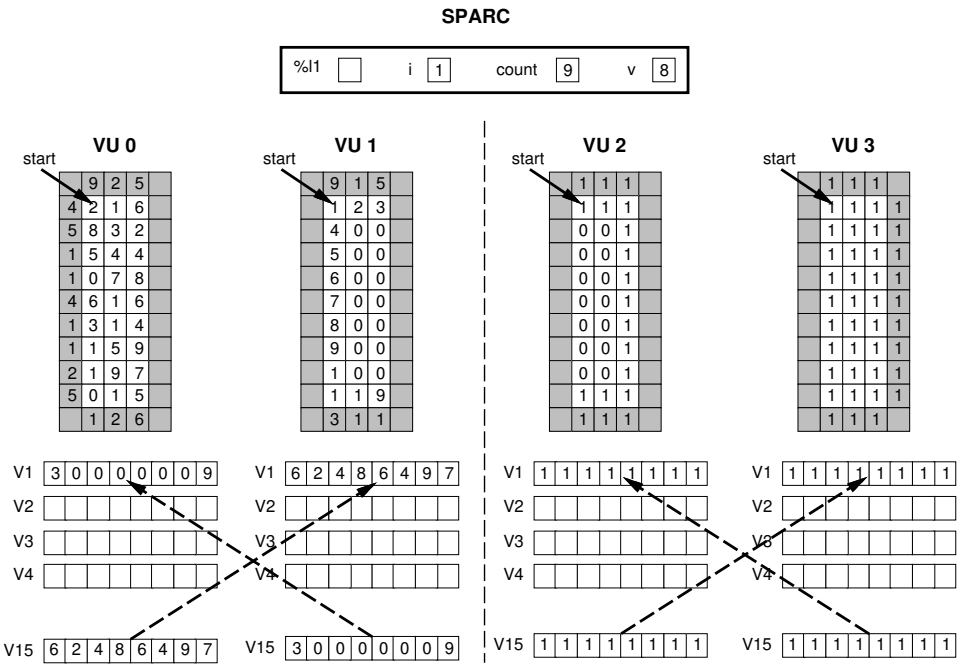

Figure 16: Next we exchange values between each VU's V15 and its partner (on the same board). The values end up in V1. Note that we have now satisfied VU1 and VU3; they have their vector of left neighbours in V1

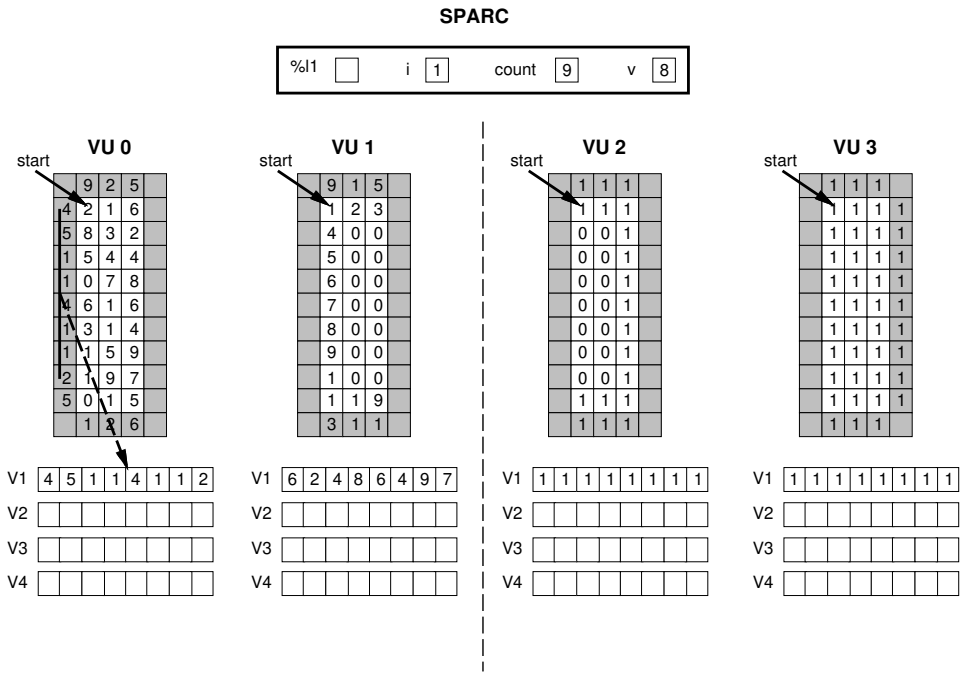

Figure 17: We overwrite the incorrect values in VU0's V1 registers with the correct values from its border region

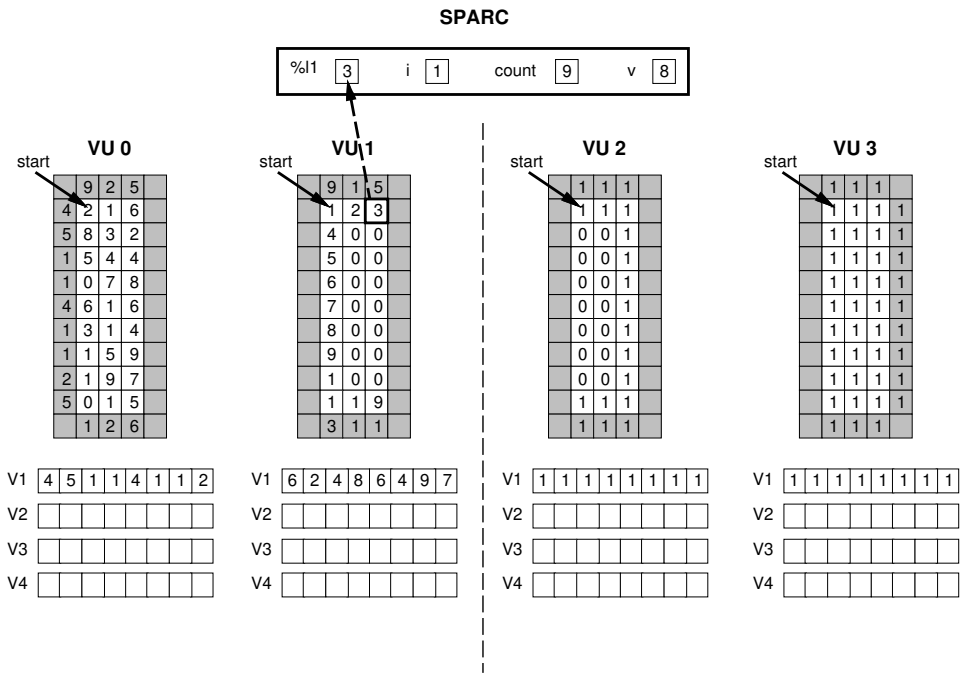

Figure 18: To overwrite the incorrect values in VU2's registers we must elementwise transfer values through the SPARC. Here the first value is read from VU1's memory to a SPARC register

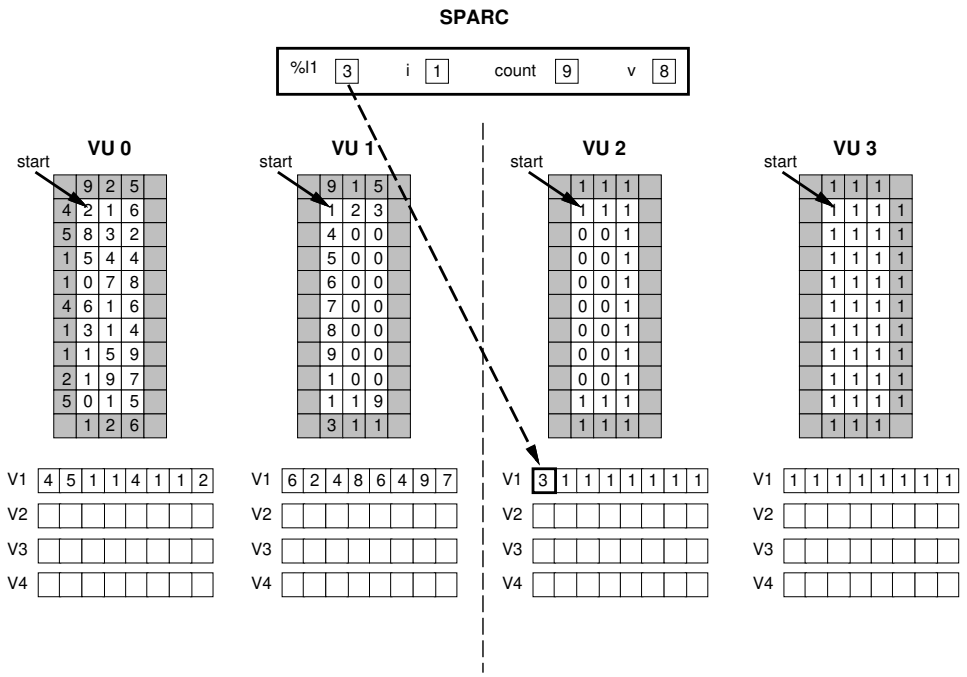

Figure 19: The value from the SPARC register is written to VU2's register overwriting the incorrect value placed there by the earlier exchange

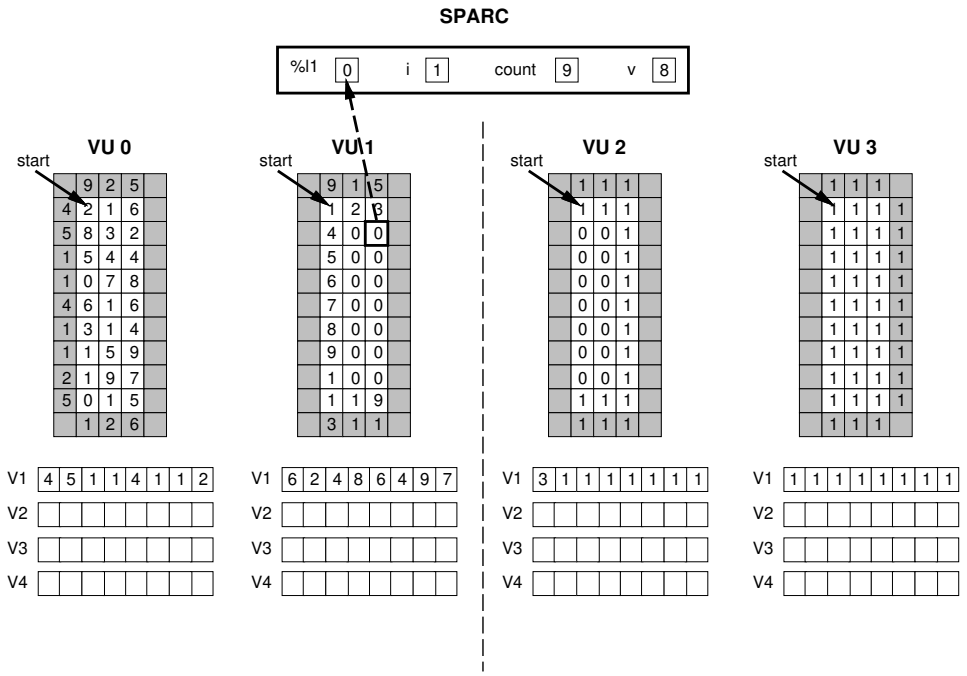

Figure 20: The second value to be transferred from VU1 to VU2 is read to SPARC register

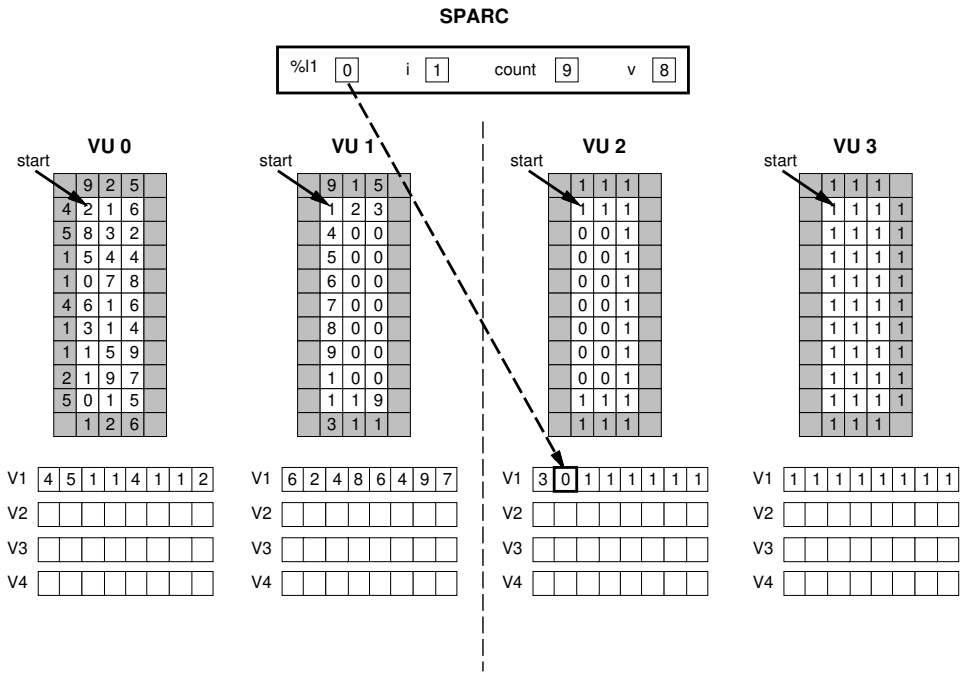

Figure 21: The second value is copied from the SPARC register to VU2's register. A further series of such copies to the SPARC and from the SPARC takes place to transfer the rest of the values

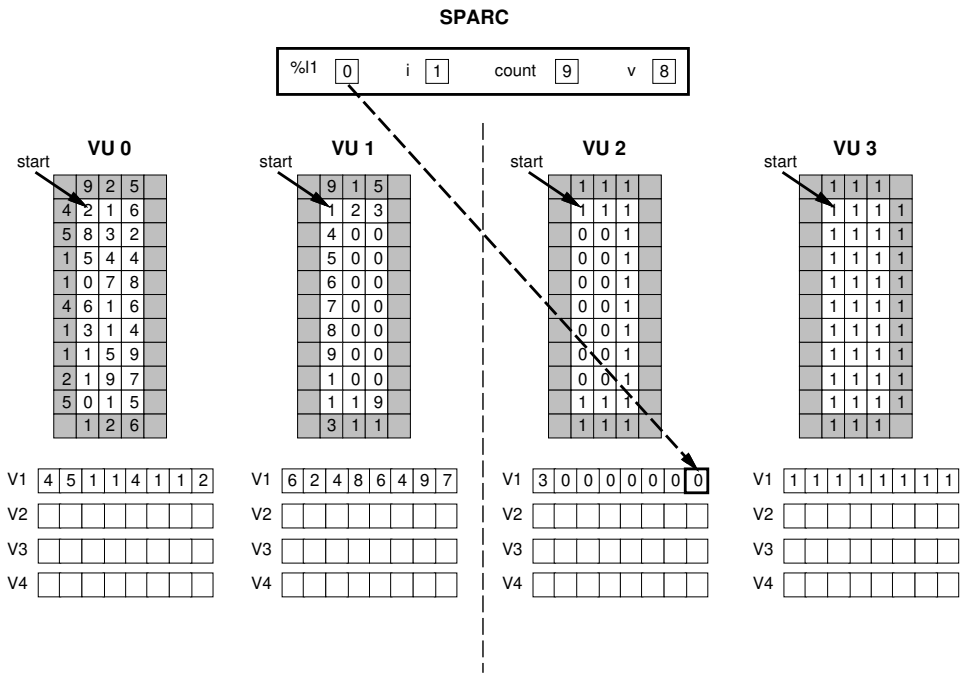

Figure 22: The final value is copied from the SPARC register to VU2's register

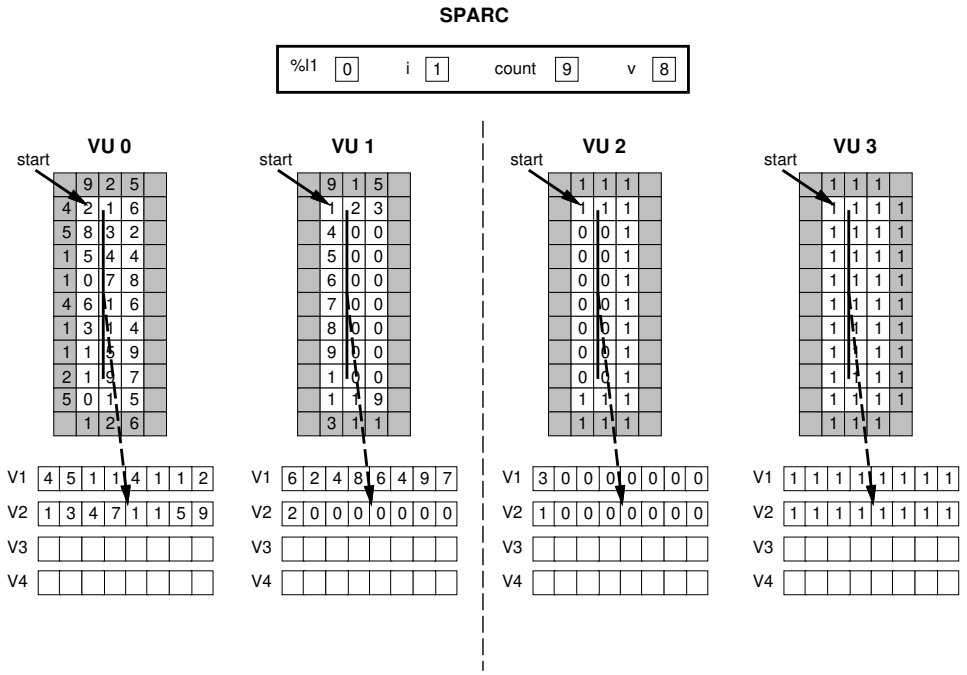

Figure 23: The vector of right neighbours is fetched into each VU's V2 registers. Since we are not looking at the last column of the subgrid this is just a simple vector load

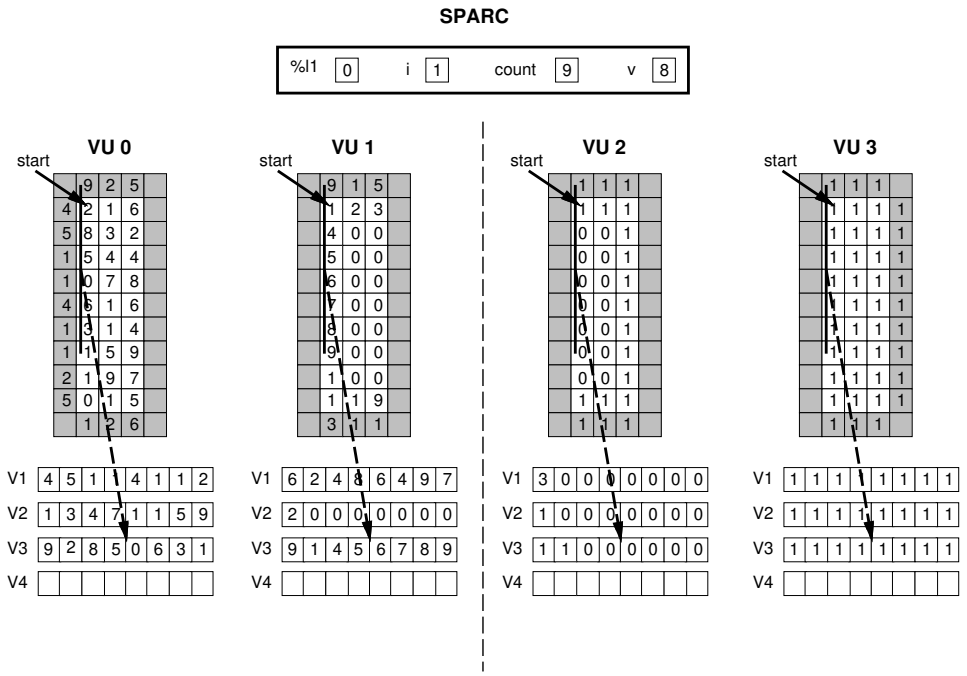

Figure 24: The vector of above neighbours is fetched into each VU's V3 registers

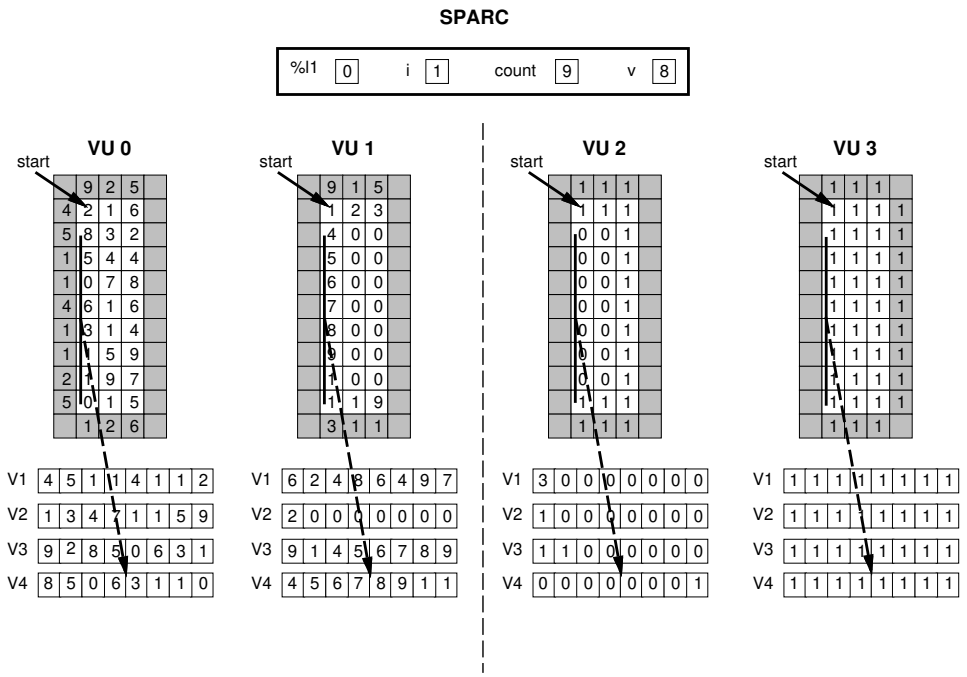

Figure 25: The vector of below neighbours is fetched into each VU's V4 registers. Once this is done we have all the data necessary to compute a vector of solutions. We add the four vectors V1, V2, V3 and V4 elementwise and write the result vector to a subgrid copy

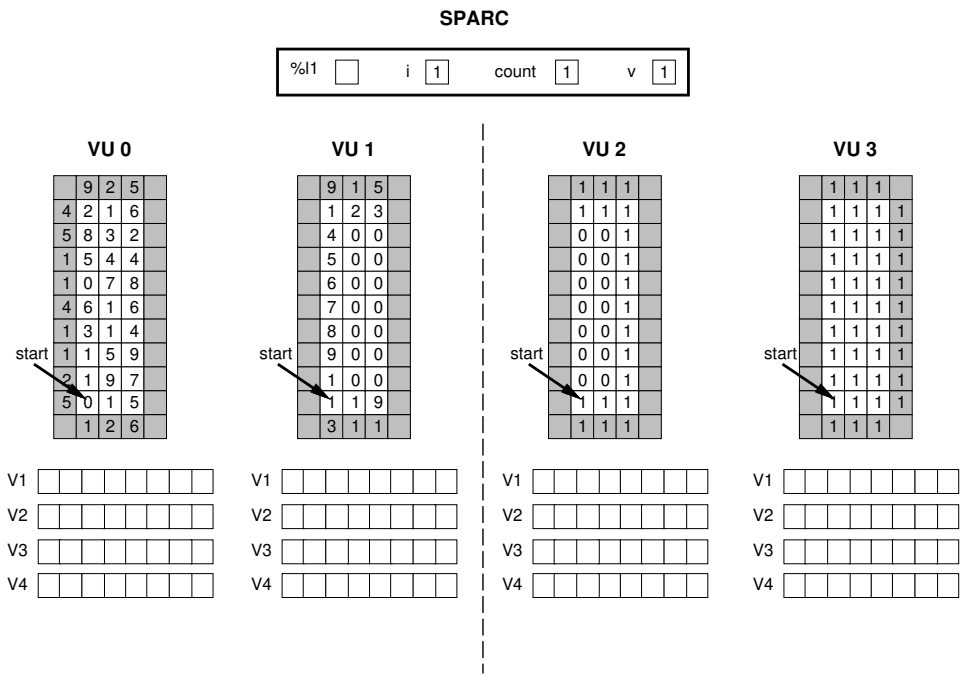

Figure 26: Now that we have completely processed 8 elements of column 1, we decrement `count` by 8 and compute a new vector length based on that new value. We set `start` to point at the first element to be processed in the next pass over the column

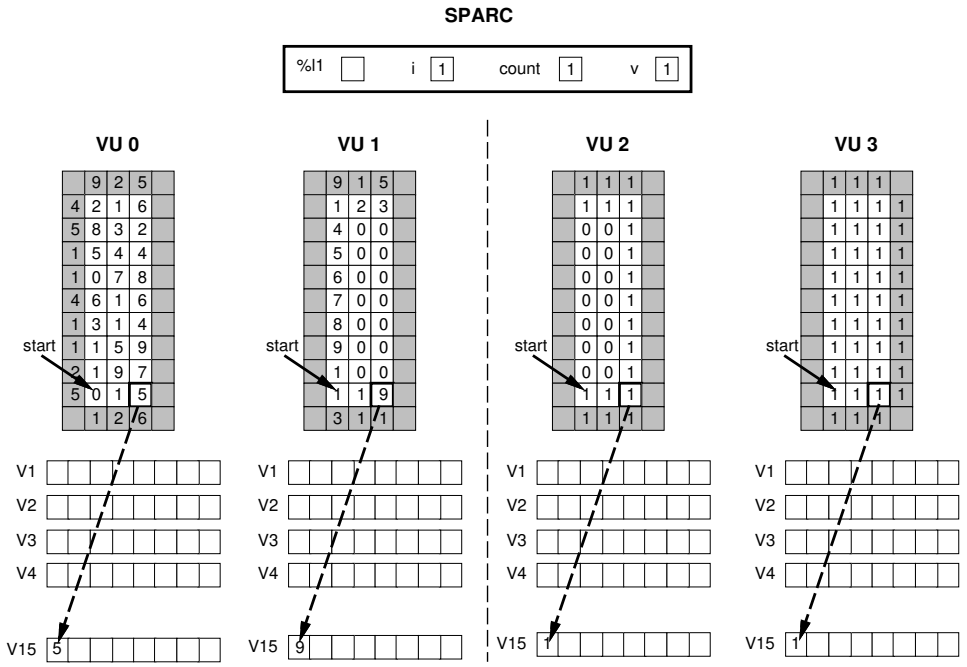

Figure 27: Once again we need to load left neighbours into V1 on each VU. Once again we need to go through a complex process since data motion between VUs is required to achieve this. We begin, as before, with loading into a temporary register V15. N.B., the vector length is now set to 1

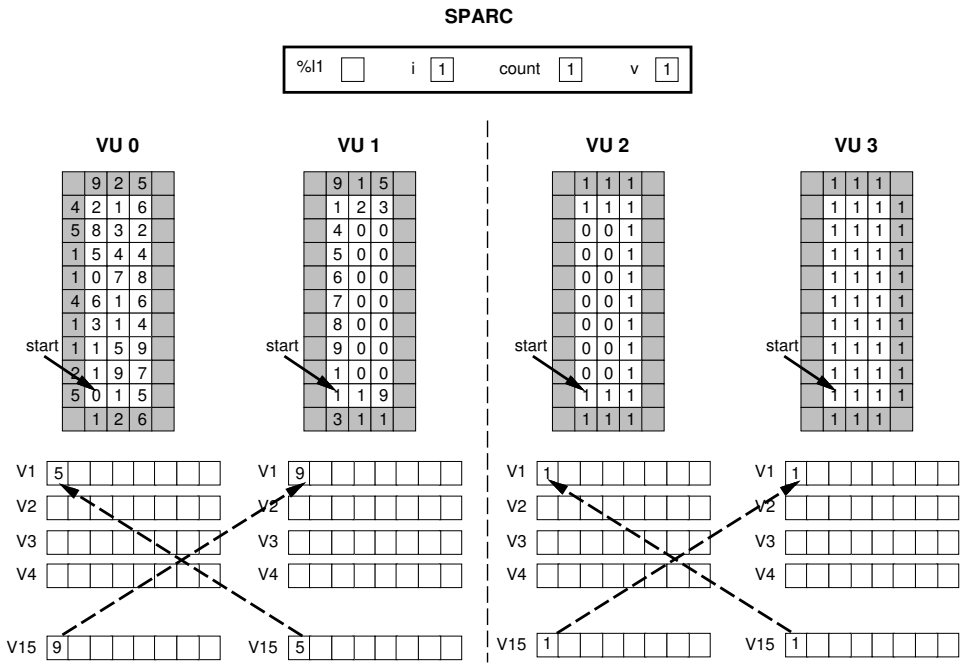

Figure 28: As before we exchange values between VUs sharing boards. This leads to VUs 1 and 3 getting correct values in their V1 registers

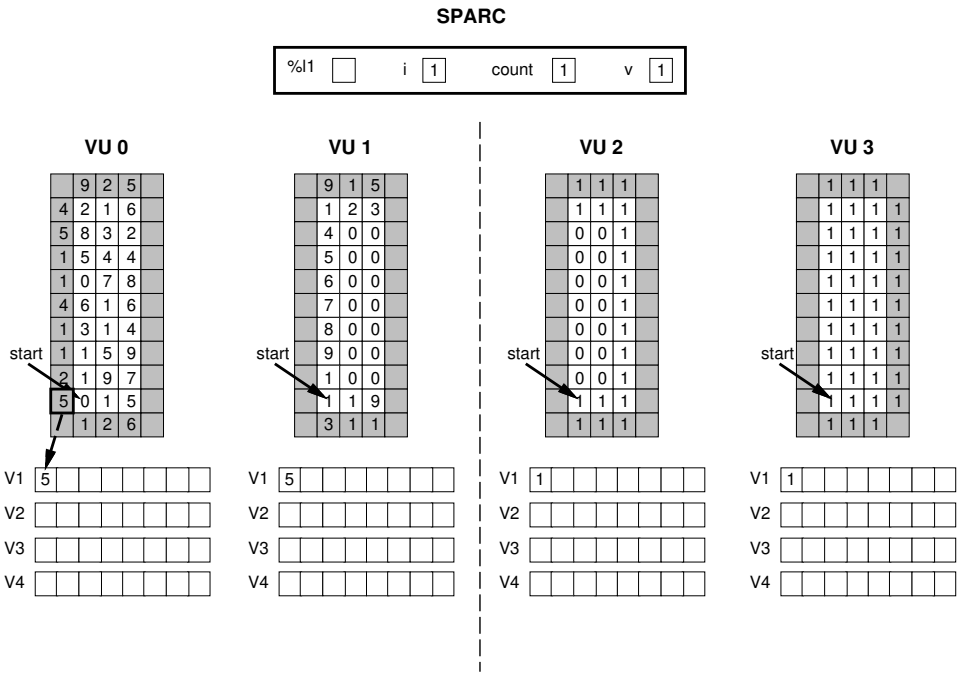

Figure 29: Again we overwrite the incorrect value in VU0's registers by loading from its border region

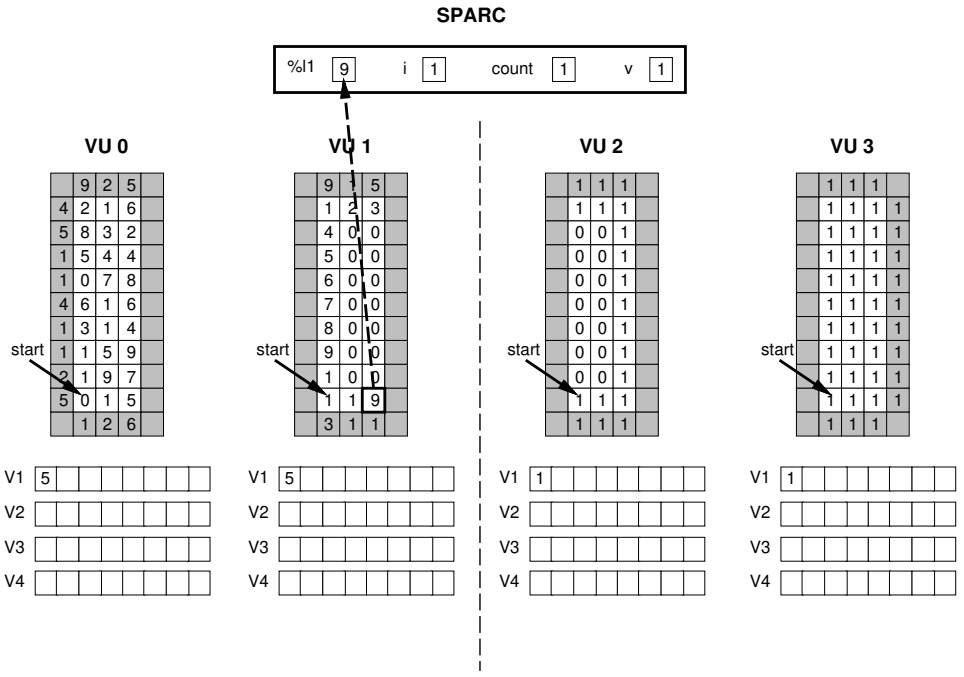

Figure 30: We need to transfer a single value from VU1's memory to VU2's registers. This requires a two step process involving the SPARC. Firstly we load the value from VU1's memory to a SPARC register

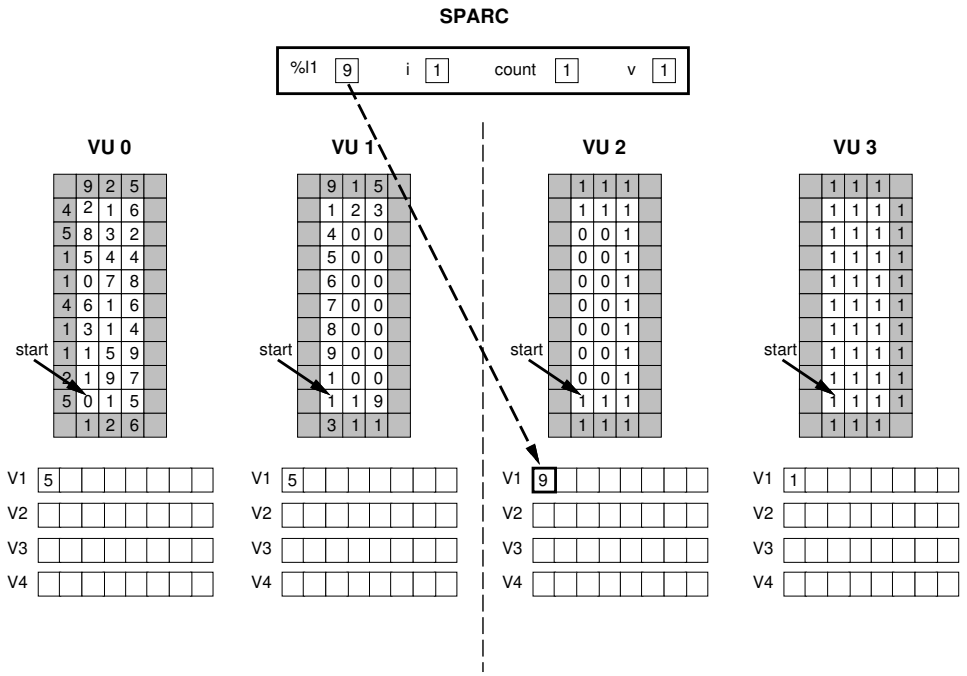

Figure 31: We then write from the SPARC register to VU2's register

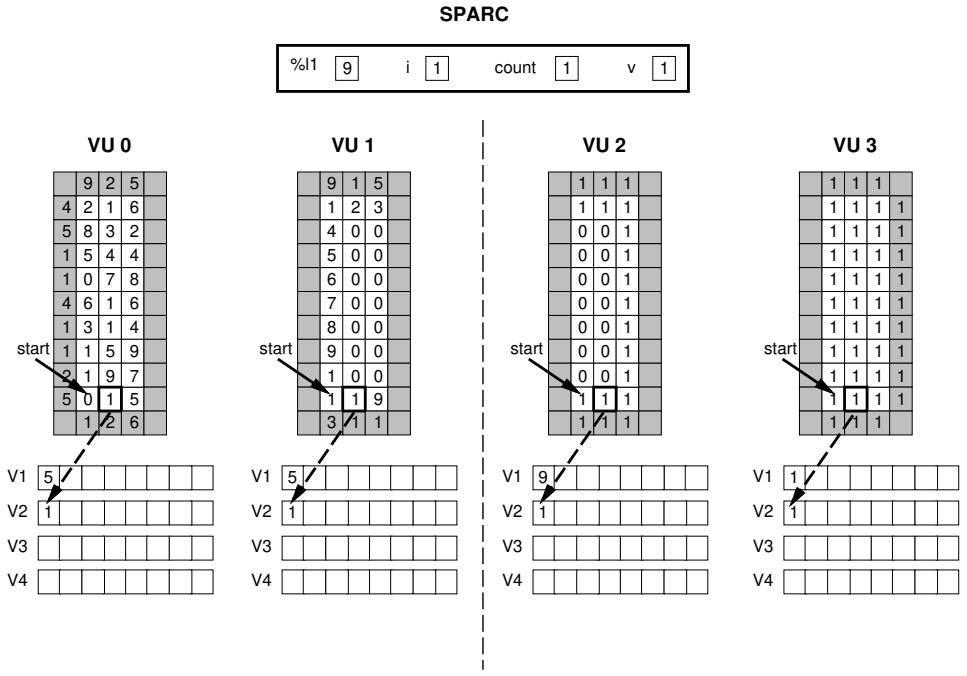

Figure 32: We load the vector of right neighbours (actually only 1, since our vector length is 1) into each VU's V2 registers

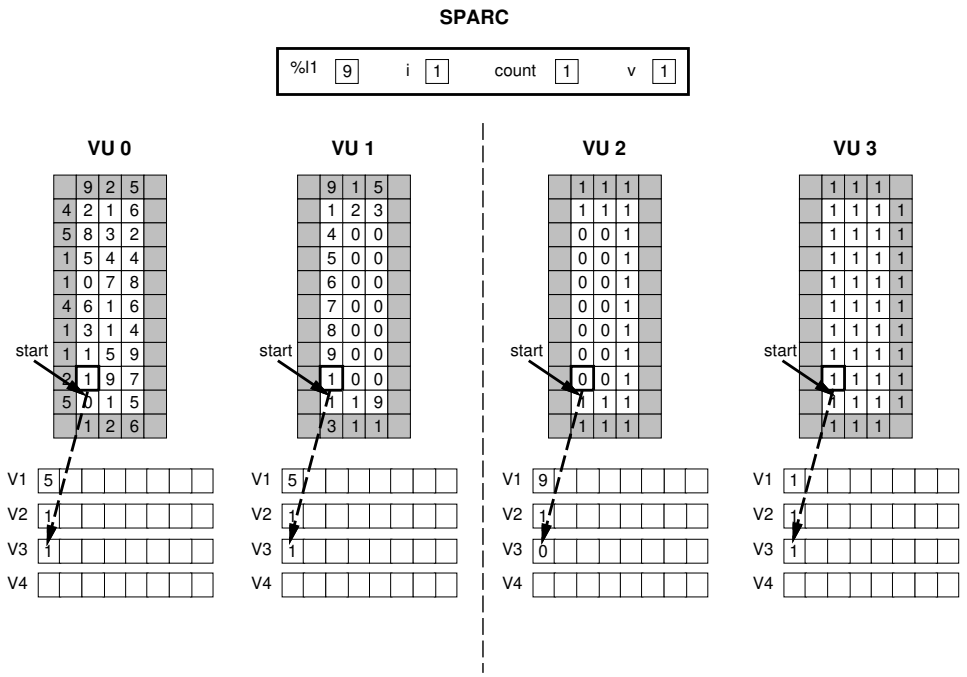

Figure 33: We load the vector of above neighbours into each VU's V3 registers

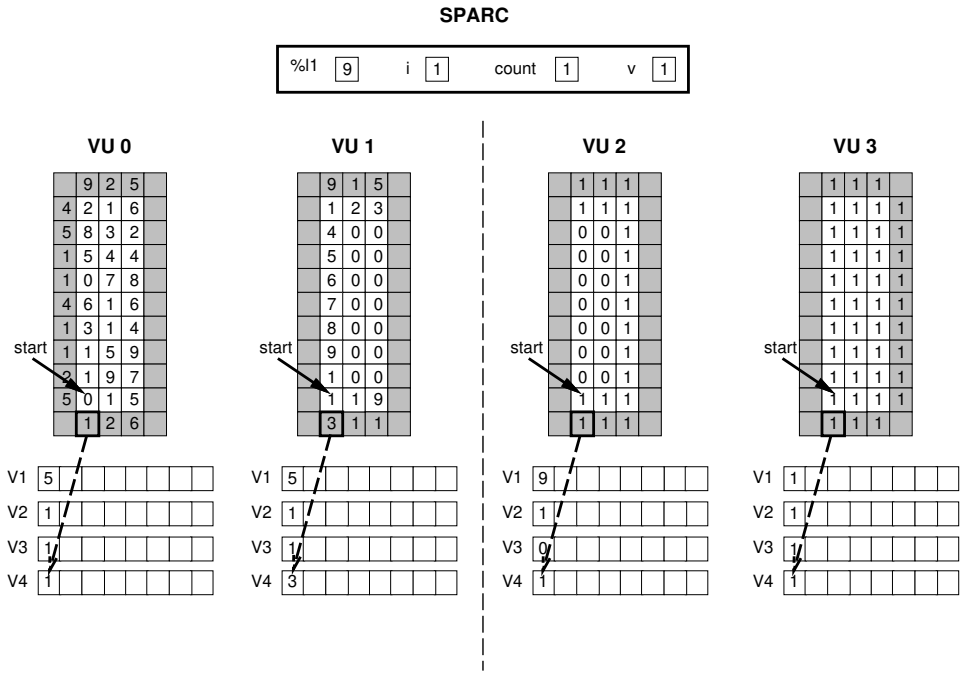

Figure 34: We load the vector of below neighbours into each VU's V3 registers. We now have all the data necessary for each VU to compute another vector of solutions. Each VU elementwise adds vectors and divides the result by the constant 4.0. The solution vector is written to a copy of the subgrid. We have now completely processed an entire column. We would then proceed to consider column 2