**II- Kernel Fusion**

The optimization by kernel fusion is summarized to the following components:

First, gathering metadata about the input of original kernels: kernels’ performances and characteristics, the data dependency graph and kernel order-of-execution graph. Second, define a performance projection model that depends on the metadata from step one and architecture features. The performance model should be lightweight, i.e., requiring no intermediate step to generate some form of code representation. Third, introduce a search algorithm that searches the space of possible fusions. Finally, use the search algorithm to generate the new fused kernels. Fig. 1 shows a dependency graph for RK routine in SCALE-LES weather application [1]. RK routine is calculated to have almost 50% redundancy, i.e., the number of loads and stores from GPU device memory would be halved due to data caching in on-chip shared memory. Fig. 2 shows an extract from the RK code that illustrates how the original kernels are transformed to fused kernels by rearranging the kernels, fusing them, then caching the shared arrays.

**III- Kernel Fusion as an Optimization Problem**

Kernel fusion is represented as a variant of the bin packing problem. In the bin packing problem, objects of different volumes must be packed into a finite number of bins or containers each of a pre-defined volume in a way that maximizes the number of bins used. The problem is hand in hand, similar; minimize the total amount of occupied bins by fusing kernels of different performances into kernels that have a total runtime less than the original runtime.

**IV- Performance Projection Model**

To have a scalable kernel fusion, a codeless projection of the bound of performance for fused kernels at each generation in the search process is used. The performance model is based on the model for projecting bounds on performance in [2].

**V- Performance Improvement by Kernel Fusion**

Algorithm 1 shows the steps for improving performance by kernel fusion. Step 4 is the projection model used as an objective function for the search heuristic. Steps 3 to 8 show the stochastic search method used. Step 9, manual fusion, can be automated.

**VI- Results**

Real-world Application: SCALE-LES

SCALE-LES is a next-generation weather model designed to configure the effects of 3D grid spacing, domain size, aspect ratio and precision on the model accuracy. SCALE-LES [1] includes over a hundred kernel of which most are memory-bound. Fusion efficiency (FE) compares the reduction in runtime of kernel F to reduction in number of memory operations due to fusion:

\[ FE = \frac{D_F + St_F}{D_F + St_F + T(F)} \]

The model implicitly deduces the practical performance bound depending on CUDA runtime’s ability of hiding the latency in a specific kernel via register and SMEM blocking factors:

\[ Block_{SMEM} = Thr + B = \text{Rem} \]

For RegFrac the register reuse factor for stencil computation, \(H_P = \text{Thr} \times B\) is the registers required for halo layer(s) divided evenly on threads per block. The value c is equal to one if halo layer(s) was used in the fused kernel to maintain coherence, otherwise equal to zero.

\[ \text{RegFrac} = \max(\text{OrdStr}(B) + c \times H_P + 1 \times \text{Rem}) \leq \text{Rem}_{\text{Max}} \]

For \( B_{\text{rem}} = B_{\text{rem}} \times (1 + \text{OrdStr}(B) + c \times H_P + 1 \times B) \) being the extra registers to fetch data from GMEM to SMEM, to accommodate the register capacity constrain in equation 1.7.

\[ R_{\text{total}} + \text{RegFrac} \times \text{OrdStr}(B) + c \times H_P + 1 \times \text{rem} \leq \text{Rem}_{\text{Max}} \]

To accommodate the SMEM capacity constrain in equation 1.6:

\[ (1 + c \times H_P) + B_{\text{rem}} + Block_{SMEM} \geq S_{\text{SMEM}} \]

The SMEM blocking factor \( B_{\text{rem}} \) is a direct function of the number of active threads, halo size and the number of shared arrays.

\[ B_{\text{rem}} = B_{\text{rem}} + (1 + \text{OrdStr}(B) + c \times H_P) \times S_{\text{SMEM}} \]

The SMEM blocking factor \( B_{\text{rem}} \) is a direct function of the number of active threads, halo size and the number of shared arrays.

\[ B_{\text{rem}} = B_{\text{rem}} + (1 + \text{OrdStr}(B) + c \times H_P) \times S_{\text{SMEM}} \]

For \( B_{\text{rem}} = B_{\text{rem}} + \text{SMEM}/(Thr \times B) \), the performance bounded by GMEM BW is estimated to be:

\[ \text{Band}_{\text{Total}} = \frac{B_{\text{rem}}}{\text{GMEM BW}} \times \text{GLOPS} \]

If \( N \) is the set of original kernels fused to the current fused kernel, \( M \) is the set of original kernels that were fused into the current kernel and have to compute a halo layer(s) and \( S \) is the set of data arrays that required a halo layer(s). The projected runtime (Sec) of the fused kernel would be:

\[ \text{Tsec} = \sum_{F} \left( \frac{f_{\text{tot}} - f_{\text{sec}}}{\text{Band}_{\text{Total}}} \right) \]

Where \( f_{\text{tot}} \) is the total execution time of the fused kernel, \( f_{\text{sec}} \) is the time to compute a halo layer(s)