* What optimizations can be applied to improve performance for different parallel platforms and
* How the OP2 framework facilitate the deployment of such optimizations

OP2 targets the domain of unstructured mesh problems and follows the design of an active library using source-to-source translation and compilation to generate multiple parallel implementations, which can then be linked against the appropriate parallel library (e.g. OpenMP, CUDA, MPI, OpenCL etc.).

Furthermore, unlike structured meshes, which utilize a regular stencil, un- structured mesh based solutions use the explicit connectivity between elements during computation: (1) sets, (2) data on sets, (3) connectivity (or mapping) between the sets and (4) operations over sets.

Associated with these sets are data (e.g. node coordinates, edge weights, velocities) and mappings between sets defining how elements of one set connect with the elements of another set.

The calls to op\_decl\_set, op\_decl\_map and op\_decl\_dat give OP2 full ownership of mappings and the data. OP2 holds them internally as C arrays and it is able to apply optimizing transformations in how the data is held in memory. Transformations include reordering mesh elements partitioning (under MPI) and conversion to an array-of-structs data. These transformations, and OP2’s ability to seamlessly apply them internally is key to achieving a number of performance optimizations.

To avoid race conditions due to indirectly accessed data the blocks are colored such that adjacent blocks are given different colors. When executing the computations per block, only blocks of the same color are executed in parallel.

We also experimented by using OpenMP’s static and dynamic load balancing functionality but did not obtain any significant benefits as all the blocks executed (except for the very last one) have the same size.

In MPI, As with all message passing based parallelizations, one of the main problems that limits the scalability is the over-partitioning of the mesh at higher machine scale. This leads to an increase in redundant computation at the halo regions (compared to the non-halo elements per partition) and an increase in time spent during halo exchanges.

We expected MPI+OpenMP to perform better at larger machine scales as observed in previous performance studies using the Airfoil CFD benchmark. The reason was that larger partition sizes per MPI process gained with MPI+OpenMP in turn resulting in smaller proportionate halo sizes. But for Hydra, adding OpenMP multi-threading has caused a reduction in performance, where the gains from better halo sizes at increasing scale have not manifested into an overall performance improvement.

**Size of the block:**

The size of the block determines the amount of work that a given thread carries out uninterrupted. The bigger it is, the higher data reuse within the block with better cache and prefetch engine utilization. At the same time, some parallelism is lost due to the colored execution. In other words, only those blocks that have the same color can be executed at the same time by different threads, with an implicit synchronization step between colors. This makes the execution scheme prone to load imbalances, especially when the number of blocks with a given color is comparable to the number of threads that are available to execute them.

As OP2 does a colored execution, and the colors are dynamically determined at runtime and may vary between different loops, statically determining which thread processes which blocks is not possible, thus allocating memory close to the thread/core that will execute a block is not feasible. Thus, the pure OpenMP version gets affected, preventing further performance gains from increased block size. When executing in an MPI+OpenMP hybrid setting, processes and threads are pinned to specific sockets, thereby circumventing this issue. Thus it is important to use a sensible MPI and OpenMP process and thread combination for the given node/socket architecture.

If the partition size is a multiple of the block size, then each thread applies the kernel on multiple set elements. Finally, if the block size is a multiple of the partition size, then a section of the threads in a block is not used, leaving executing threads more resources.