ScalFmm – C Kernel API

- To be able to implement a FMM kernel without knowing the C++ or ScalFmm
- Delegate the containers (data management) and the FMM algorithm to ScalFmm
- The user needs to implement the usual operators (P2M/M2M/M2L/L2L/L2P)
- A small example is given in Addons/CKernelAPI/Tests
- Available in the next release (15 September 2014)

Initalization and inserting the particles

- Get a ScalFmm handler :

  Scalfmm_Handle handle = Scalfmm_init_handle(treeHeight, boxWidth, boxCenter);

- Insert the particles :

  Scalfmm_insert_array_of_particles(handle, nbParticles, particleIndexes, particleXYZ);

  - ParticlesIndex (array : integer × nbParticles)
  - ParticlesXYZ (array : double × 3 × nbParticles)

- The particle indexes should be used to retreive the information of the particles

Setting the kernel Callbacks

- The user must fill the structure :

```
struct Scalfmm_Kernel_Descriptor {
    Callback_P2M p2m;
    Callback_M2M m2m;
    Callback_M2L m2l;
    Callback_L2L l2l;
    Callback_L2P l2p;
    Callback_P2P p2p;
    Callback_P2PInner p2pinner;
};
```

- And call :

```
Scalfmm_execute_kernel(handle, kernel, &my_data);
```

Callback Example : the M2M

- The user's M2M can look like :

```
void my_Callback_M2M(int level, void* cellData, int childPosition, void* childData, void* userData){
    struct MyData* my_data = (struct MyData*)userData;
    struct MyCellDescriptor* my_cell = (struct MyCellDescriptor*) cellData;
    struct MyCellDescriptor* my_child = (struct MyCellDescriptor*) childData;
    // JUST-PUT-HERE: your real M2M computation between parent and child
}
```

-

Callback Example : the P2P

- The user's P2P between two leaves could be :

```
void my_Callback_P2P(int nbParticlesInLeaf, const int* particleIndexes, int nbParticlesInSrc, const int*
particleIndexesSrc, void* userData){

    struct MyData* my_data = (struct MyData*)userData;

    int idxPartTarget, idxPartSource;

    for(idxPartTarget = 0 ; idxPartTarget < nbParticlesInLeaf ; ++idxPartTarget){

        for(idxPartSource = 0 ; idxPartSource < nbParticlesInSrc ; ++idxPartSource){

            // Compute your P2P between 2 particles of your indexes particleIndexes[idxPartTarget]

            // and particleIndexesSrc[idxPartSource] (and your data using userData parameter)

        }

    }

}
```

-

Perspective

- Supporting MPI parallelization using the API (only OpenMP for now)
- Proposing some methods to reset some data or iterate in the tree
-
- Refer to the example and the API header (or the doxygen doc) to know more