Software Requirements Specification

for

<Mesh-based Application>

Version 1.0 approved

Prepared by <陈灿辉 17343008>

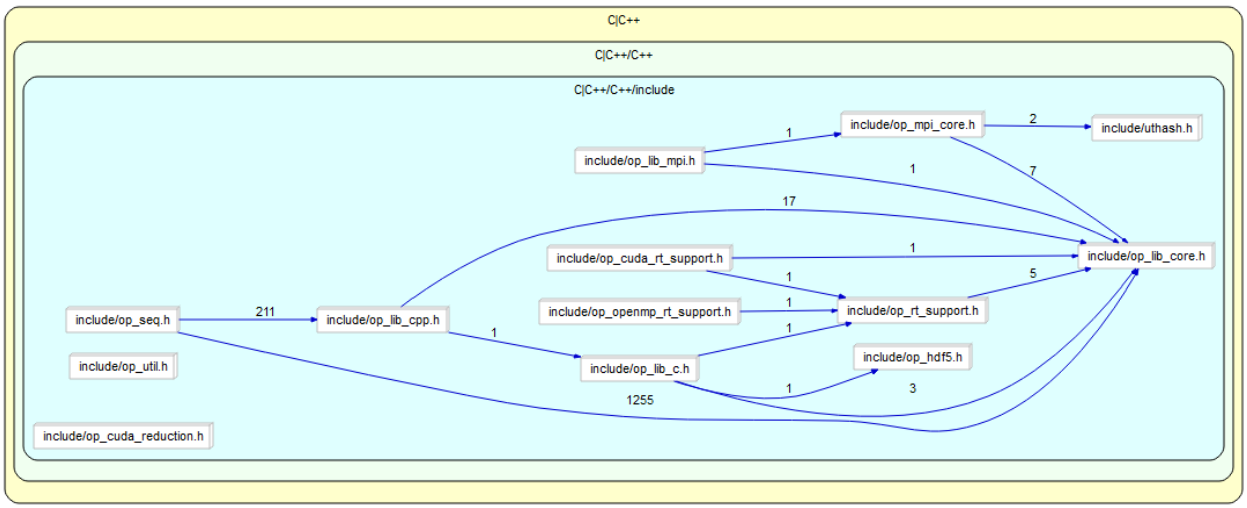
<中山大学数据科学与计算机学院软工一班>

<2019-05-09>

Study the Op2 source code starting from *.h files in folder \OP2\op2\c\include (op_lib_core.h, op_mpi_core.h, op_rt_support.h, op_seq.h). Then try to find some main data structures (set, map, dat, arg, kernel, plan) and their related operations (functions) from those and related source files. Write a specification about these data structures and their related operations. In your specification, try to use abstractions you think which can describe the domain understanding of these data structures and operations. (Please refer to the doc folder for references. Try to understand Some terminologies from **the field of mesh-based applications**. Your specification may be considered as the initialization of domain conceptual modeling.)

OP2代码框架

我们首先考虑 \OP2\op2\c\include*.h 的所有文件，由此可见各文件的include关系如下图所示。

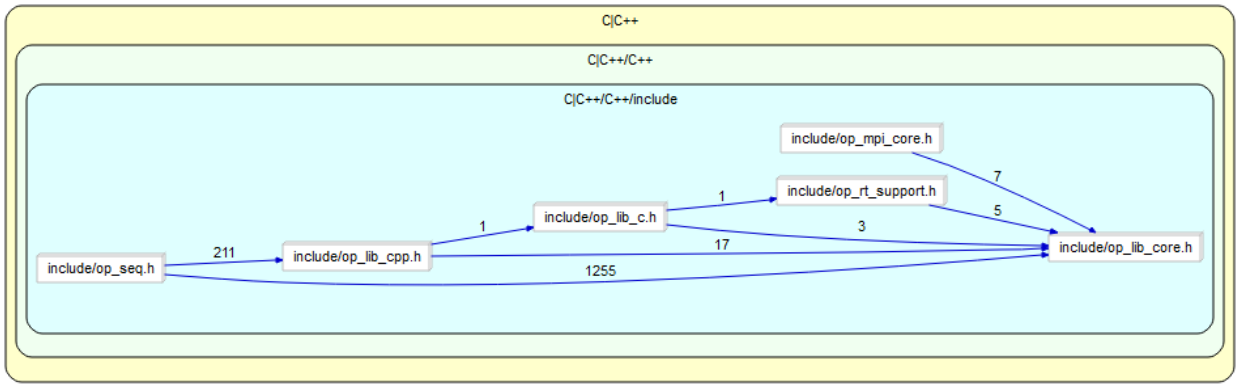


下面我们重点考察op_lib_core.h, op_mpi_core.h, op_rt_support.h, op_seq.h

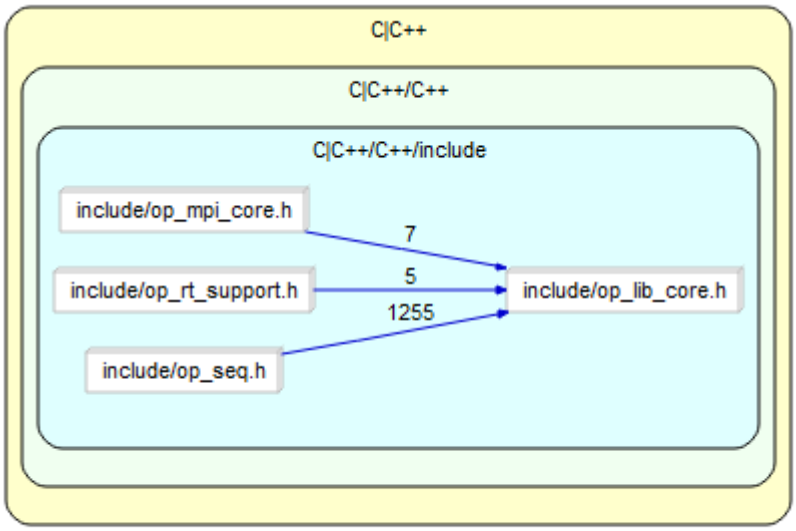
对应文件及其功能说明

文件名	功能
op_lib_core.h	This header file declares all types and functions required by *any* OP2 implementation, i.e. independently of the back-end and the target languages. It is typically used by language or back-end specific top level OP2 libraries
op_mpi_core.h	Headder file for the OP2 Distributed memory (MPI) halo creation, halo exchange and support utility routines/functions
op_rt_support.h	This header file defines the data structures required in the OP2 run-time support, i.e. those related to OP2 plans, and it declares the low-level routines used to build OP2 plans
op_seq.h	header for sequential and MPI+sequentional execution

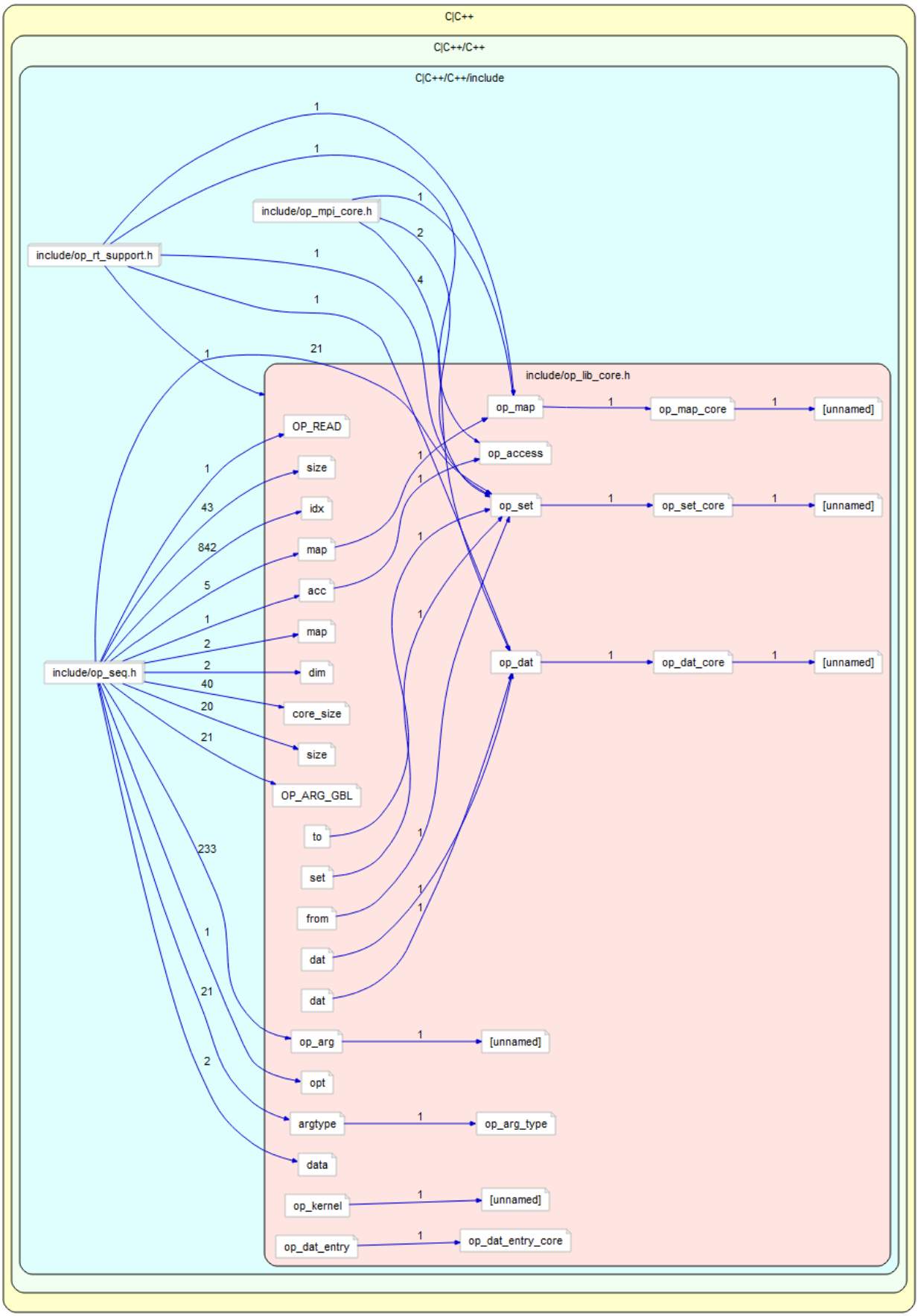
这4个头文件之间的关系，简化后关系如图所示



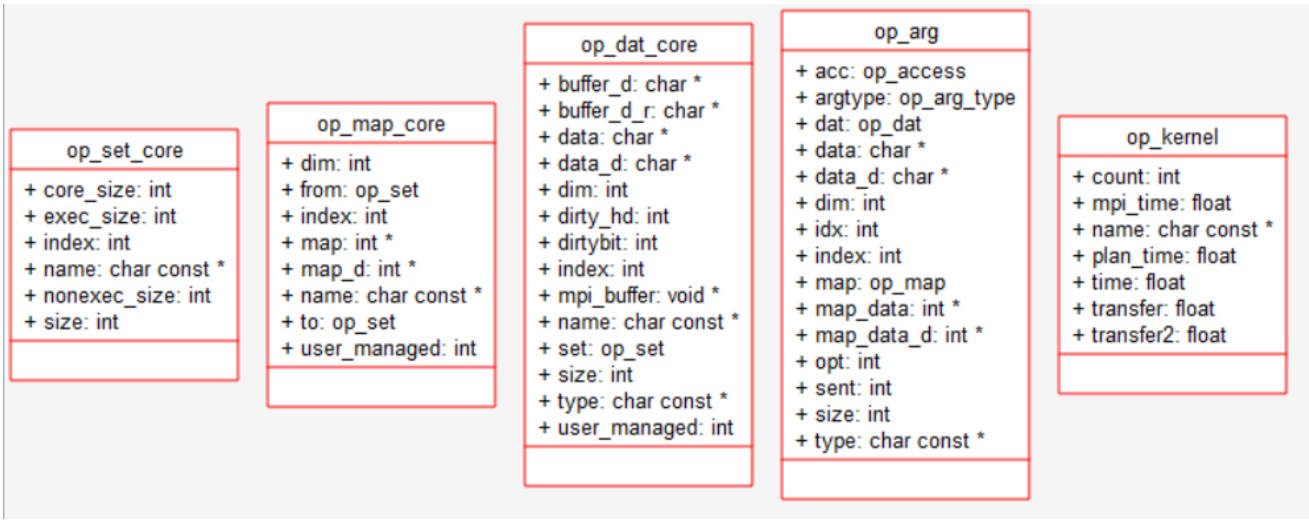
进一步简化，可以看出依赖关系大致是一个树形，其中树根为op_lib_core.h, 也能看出其为整个项目的核心文件



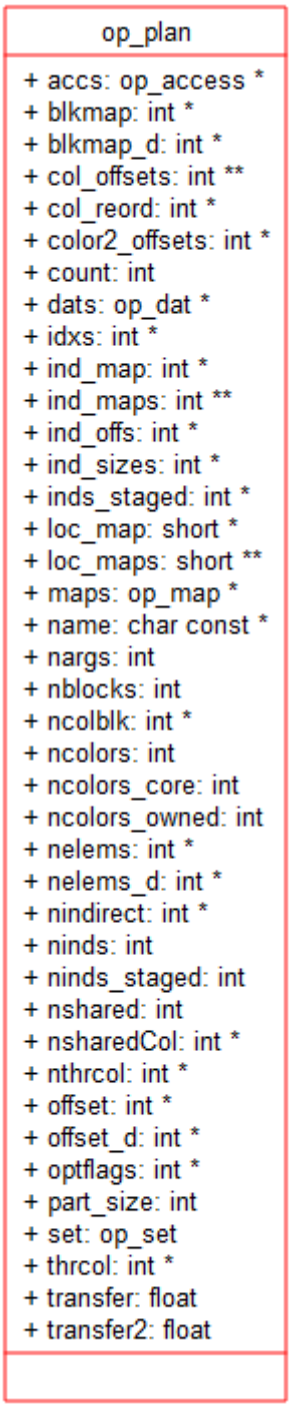
op_mpi_core.h, op_rt_support.h, op_seq.h 中具体的调用关系如下



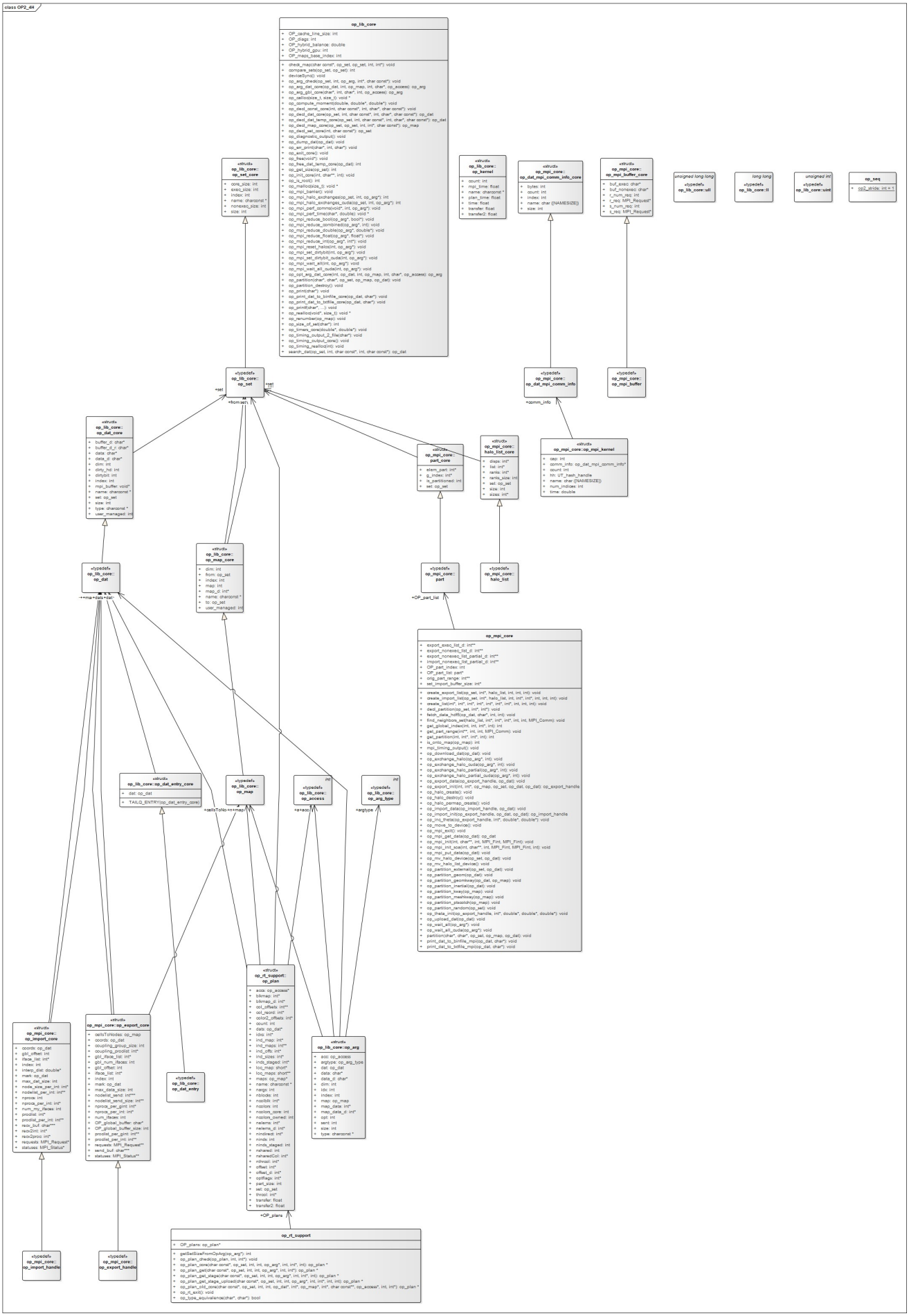
op_lib_core.h中关键成员的UML视图如下



在op_rt_support.h中 op_plan 的UML视图如下所示



通过逆向工程，分析代码得到四个头文件之间的关系入下图所示


class OP2_4H
op_lib_core
+ OP_cache_line_size: int
+ OP_diags: int
+ OP_hybrid_balance: double
+ OP_hybrid_gpu: int
+ OP_maps_base_index: int
+ check_map(char const*, op_set, op_set, int, int*): void
+ compare_sets(op_set, op_set): int
+ deviceSync(): void
+ op_arg_check(op_set, int, op_arg, int*, char const*): void
+ op_arg_dat_core(op_dat, int, op_map, int, char*, op_access): op_arg
+ op_arg_gbl_core(char*, int, char*, int, op_access): op_arg
+ op_calloc(size_t, size_t): void *
+ op_compute_moment(double, double*, double*): void
+ op_decl_const_core(int, char const*, int, char*, char const*): void
+ op_decl_dat_core(op_set, int, char const*, int, char*, char const*): op_dat
+ op_decl_dat_temp_core(op_set, int, char const*, int, char*, char const*): op_dat
+ op_decl_map_core(op_set, op_set, int, int*, char const*): op_map
+ op_decl_set_core(int, char const*): op_set
+ op_diagnostic_output(): void
+ op_dump_dat(op_dat): void
+ op_err_print(char*, int, char*): void
+ op_exit_core(): void
+ op_free(void*): void
+ op_free_dat_temp_core(op_dat): int
+ op_get_size(op_set): int
+ op_init_core(int, char**, int): void
+ op_is_root(): int
+ op_malloc(size_t): void *
+ op_mpi_barrier(): void
+ op_mpi_halo_exchanges(op_set, int, op_arg*): int
+ op_mpi_halo_exchanges_cuda(op_set, int, op_arg*): int
+ op_mpi_perf_comms(void*, int, op_arg*): void
+ op_mpi_perf_time(char*, double): void *
+ op_mpi_reduce_bool(op_arg*, bool*): void
+ op_mpi_reduce_combined(op_arg*, int): void
+ op_mpi_reduce_double(op_arg*, double*): void
+ op_mpi_reduce_float(op_arg*, float*): void
+ op_mpi_reduce_int(op_arg*, int*): void
+ op_mpi_reset_halos(int, op_arg*): void
+ op_mpi_set_dirtybit(int, op_arg*): void
+ op_mpi_set_dirtybit_cuda(int, op_arg*): void
+ op_mpi_wait_all(int, op_arg*): void
+ op_mpi_wait_all_cuda(int, op_arg*): void
+ op_opt_arg_dat_core(int, op_dat, int, op_map, int, char*, op_access): op_arg
+ op_partition(char*, char*, op_set, op_map, op_dat): void
+ op_partition_destroy(): void
+ op_print(char*): void
+ op_print_dat_to_binfile_core(op_dat, char*): void
+ op_print_dat_to_txtfile_core(op_dat, char*): void
+ op_printf(char*, ...): void
+ op_realloc(void*, size_t): void *
+ op_renumber(op_map): void
+ op_size_of_set(char*): int
+ op_timers_core(double*, double*): void
+ op_timing_output_2_File(char*): void
+ op_timing_output_core(): void
+ op_timing_realloc(int): void
+ search_dat(op_set, int, char const*, int, char const*): op_dat
«struct» op_lib_core::op_set_core
+ core_size: int
+ exec_size: int
+ index: int
+ name: charconst *
+ nonexec_size: int
+ size: int
«struct» op_lib_core::op_kernel
+ count: int
+ mpi_time: float
+ name: charconst *
+ plan_time: float
+ time: float
+ transfer: float
+ transfer2: float
«struct» op_mpi_core::op_dat_mpi_comm_info_core
+ bytes: int
+ count: int
+ index: int
+ name: char [NAMESIZE]
+ size: int
«struct» op_mpi_core::op_mpi_buffer_core
+ buf_exec: char*
+ buf_nonexec: char*
+ r_num_req: int
+ r_req: MPI_Request*
+ s_num_req: int
+ s_req: MPI_Request*
unsigned long long
«typedef» op_lib_core::ull
long long
«typedef» op_lib_core::ll
unsigned int
«typedef» op_lib_core::uint
op_seq
+ op2_strds: int = 1
«typedef» op_lib_core::op_set
«typedef» op_mpi_core::op_dat_mpi_comm_info
«typedef» op_mpi_core::op_mpi_buffer
«struct» op_lib_core::op_dat_core
+ buffer_d: char*
+ buffer_d_r: char*
+ data: char*
+ data_d: char*
+ dim: int
+ dirty_hd: int
+ dirtybit: int
+ index: int
+ mpi_buffer: void*
+ name: charconst *
+ set: op_set
+ size: int
+ type: charconst *
+ user_managed: int
«struct» op_mpi_core::part_core
+ elem_part: int*
+ g_index: int*
+ is_partitioned: int
+ set: op_set
«struct» op_mpi_core::halo_list_core
+ clags: int*
+ list: int*
+ ranks: int*
+ ranks_size: int
+ set: op_set
+ size: int
+ sizes: int*
«struct» op_mpi_core::op_mpi_kernel
+ cap: int
+ comm_info: op_dat_mpi_comm_info*
+ count: int
+ hh: UT_hash_handle
+ name: char [NAMESIZE]
+ num_indices: int
+ time: double
«typedef» op_lib_core::op_dat
«typedef» op_mpi_core::part
«typedef» op_mpi_core::halo_list
«struct» op_lib_core::op_map_core
+ dim: int
+ from: op_set
+ index: int
+ map: int
+ map_d: int*
+ name: charconst *
+ to: op_set
+ user_managed: int
«struct» op_lib_core::op_dat_entry_core
+ dat: op_dat
+ TAILQ_ENTRY(op_dat_entry_core)
«typedef» op_lib_core::op_map
«typedef» op_lib_core::op_access
int
«typedef» op_lib_core::op_arg_type
int
«typedef» op_lib_core::op_dat_entry
«struct» op_mpi_core::op_import_core
+ ocords: op_dat
+ gbl_offset: int
+ ifaces_list: int*
+ index: int
+ inteps_dist: double*
+ mark: op_dat
+ max_dat_size: int
+ node_size_per_int: int*
+ nodelist_per_int: int**
+ nprocs: int
+ nprocs_per_int: int*
+ num_my_ifaces: int
+ prodist: int*
+ prodist_per_int: int**
+ recv_buf: char***
+ recv2int: int*
+ recv2proc: int*
+ requests: MPI_Request**
+ statuses: MPI_Status**
«struct» op_mpi_core::op_export_core
+ cellsToNodes: op_map
+ ocords: op_dat
+ coupling_group_size: int
+ coupling_prodist: int*
+ gbl_iface_list: int*
+ gbl_num_ifaces: int
+ gbl_offset: int
+ iface_list: int*
+ index: int
+ mark: op_dat
+ max_data_size: int
+ nodelist_send: int***
+ nodelist_send_size: int**
+ nprocs_per_gint: int*
+ nprocs_per_int: int*
+ num_ifaces: int
+ OP_global_buffer: char*
+ OP_global_buffer_size: int
+ prodist_per_gint: int**
+ prodist_per_int: int**
+ requests: MPI_Request**
+ send_buf: char***
+ statuses: MPI_Status**
«struct» op_rt_support::op_plan
+ acss: op_access*
+ bitmap: int*
+ bitmap_d: int*
+ col_offsets: int**
+ col_reord: int*
+ color2_offsets: int*
+ count: int
+ dats: op_dat*
+ idxs: int*
+ ind_map: int*
+ ind_maps: int**
+ ind_offs: int*
+ ind_sizes: int*
+ inds_staged: int*
+ loc_map: short*
+ loc_maps: short**
+ maps: op_map*
+ name: charconst *
+ nargs: int
+ nblocks: int
+ nopbits: int*
+ nopbits: int
+ nopbits_core: int
+ nopbits_owned: int
+ nelems: int*
+ nelems_d: int*
+ nindirect: int*
+ ninds: int
+ ninds_staged: int
+ nshared: int
+ nshared2col: int*
+ nthrcol: int*
+ offset: int*
+ offset_d: int*
+ opflags: int*
+ part_size: int
+ set: op_set
+ thrcol: int*
+ transfer: float
+ transfer2: float
+OP_plans
«struct» op_lib_core::op_arg
+ acc: op_access
+ argtype: op_arg_type
+ dat: op_dat
+ data: char*
+ data_d: char*
+ dim: int
+ idx: int
+ index: int
+ map: op_map
+ map_data: int*
+ map_data_d: int*
+ opt: int
+ sent: int
+ size: int
+ type: charconst *
«typedef» op_mpi_core::op_import_handle
«typedef» op_mpi_core::op_export_handle
op_rt_support
+ OP_plans: op_plan*
+ getSetSizeFromOpArg(op_arg*): int
+ op_plan_check(op_plan, int, int*): void
+ op_plan_core(char const*, op_set, int, int, op_arg*, int, int*, int): op_plan *
+ op_plan_get(char const*, op_set, int, int, op_arg*, int, int*): op_plan *
+ op_plan_get_stage(char const*, op_set, int, int, op_arg*, int, int*, int): op_plan *
+ op_plan_get_stage_upload(char const*, op_set, int, int, op_arg*, int, int*, int, int): op_plan *
+ op_plan_old_core(char const*, op_set, int, int, op_dat*, int*, op_map*, int*, char const**, op_access*, int, int*): op_plan *
+ op_rt_exit(): void
+ op_type_equivalence(char*, char*): bool
op_mpi_core
+ export_exec_list_d: int**
+ export_nonexec_list_d: int**
+ export_nonexec_list_partial_d: int**
+ import_nonexec_list_partial_d: int**
+ OP_part_index: int
+ OP_part_list: part*
+ orig_part_range: int**
+ set_import_buffer_size: int*
+ create_export_list(op_set, int*, halo_list, int, int, int): void
+ create_import_list(op_set, int*, halo_list, int, int*, int, int, int, int): void
+ create_list(int*, int*, int*, int*, int*, int, int, int, int): void
+ dealloc_partition(op_set, int*, int*): void
+ fetch_data_hdf5(op_dat, char*, int, int): void
+ find_neighbors_set(halo_list, int*, int*, int, int, int, MPI_Comm): void
+ get_global_index(int, int, int*, int): int
+ get_part_range(int**, int, int, MPI_Comm): void
+ get_partition(int, int*, int*, int): int
+ is_onto_map(op_map): int
+ mpi_timing_output(): void
+ op_download_dat(op_dat): void
+ op_exchange_halo(op_arg*, int): void
+ op_exchange_halo_cuda(op_arg*, int): void
+ op_exchange_halo_partial(op_arg*, int): void
+ op_exchange_halo_partial_cuda(op_arg*, int): void
+ op_export_data(op_export_handle, op_dat): void
+ op_export_init(int, int*, op_map, op_set, op_dat, op_dat): op_export_handle
+ op_halo_create(): void
+ op_halo_destroy(): void
+ op_halo_permap_create(): void
+ op_import_data(op_import_handle, op_dat): void
+ op_import_init(op_export_handle, op_dat, op_dat): op_import_handle
+ op_inv_theta(op_export_handle, int*, double*, double*): void
+ op_move_to_device(): void
+ op_mpi_exit(): void
+ op_mpi_get_data(op_dat): op_dat
+ op_mpi_init(int, char**, int, MPI_Fint, MPI_Fint): void
+ op_mpi_init_soa(int, char**, int, MPI_Fint, MPI_Fint, int): void
+ op_mpi_put_data(op_dat): void
+ op_mv_halo_device(op_set, op_dat): void
+ op_mv_halo_list_device(): void
+ op_partition_external(op_set, op_dat): void
+ op_partition_geom(op_dat): void
+ op_partition_geomkway(op_dat, op_map): void
+ op_partition_inertial(op_dat): void
+ op_partition_kway(op_map): void
+ op_partition_meshkway(op_map): void
+ op_partition_ptscotch(op_map): void
+ op_partition_random(op_set): void
+ op_theta_init(op_export_handle, int*, double*, double*, double*): void
+ op_upload_dat(op_dat): void
+ op_wait_all(op_arg*): void
+ op_wait_all_cuda(op_arg*): void
+ partition(char*, char*, op_set, op_map, op_dat): void
+ print_dat_to_binfile_mpi(op_dat, char*): void
+ print_dat_to_txtfile_mpi(op_dat, char*): void

OP2中的相关结构

Mesh 被存放在一个 .dat 文件中，先存放一些集合（比如 nodes 、 edges 、 cells 、 bedges ）；然后再存放描述这些集合关系的映射（比如 cell->4 nodes ）。最后存放一些集合的相关数据（比如 pcell[cell][0].x[0..3]，其中 pcell[cell][0] 表示 cell 的第 0 个节点。

Mesh 的结构在OP2的存储方式（Mesh structure in Memory）

op_set_core

保存 node、edge、bedge、cell 等集合

```
typedef struct {
  int index;        /* index */
  int size;         /* number of elements in set */
  char const *name; /* name of set */
                    // for MPI support
  int core_size;    /* number of core elements in an mpi process*/
  int exec_size;    /* number of additional imported elements to be executed */
  int nonexec_size; /* number of additional imported elements that are not
                       executed */
} op_set_core;
```

op_map_core

保存一些映射关系，比如 edge->node ， edge->cell ， bedge->cell ， bedge->node ， cell->node 。这些关系是存放在指针中的

```
typedef struct {
  int index;        /* index */
  op_set from,      /* set pointed from */
      to;           /* set pointed to */
  int dim,          /* dimension of pointer */
      *map;         /* array defining pointer */
  int *map_d;       /* array on device */
  char const *name; /* name of pointer */
  int user_managed; /* indicates whether the user is managing memory */
} op_map_core;
```

op_dat_core

保存集合（ op_set ）的信息（属性名、维数、类型、已有的数据，代号）等

```
typedef struct {
  int index;        /* index */
  op_dat dat;       /* dataset */
  op_map map;       /* indirect mapping */
  int dim,          /* dimension of data */
      idx, size;    /* size (for sequential execution) */
  char *data,       /* data on host */
      *data_d;      /* data on device (for CUDA execution) */
  int *map_data,    /* data on host */
```

```
      *map_data_d;  /* data on device (for CUDA execution) */
  char const *type; /* datatype */
  op_access acc;
  op_arg_type argtype;
  int sent; /* flag to indicate if this argument has
               data in flight under non-blocking MPI comms*/
  int opt;  /* flag to indicate if this argument is in use */
} op_arg;
```

op_arg

作为传入函数的参数，每个核函数的参数都包含其名称、数据集合以及参数（用 op_arg 表示）。

```
typedef struct {
  int index;        /* index */
  op_dat dat;       /* dataset */
  op_map map;       /* indirect mapping */
  int dim,          /* dimension of data */
      idx, size;    /* size (for sequential execution) */
  char *data,       /* data on host */
      *data_d;      /* data on device (for CUDA execution) */
  int *map_data,    /* data on host */
      *map_data_d;  /* data on device (for CUDA execution) */
  char const *type; /* datatype */
  op_access acc;
  op_arg_type argtype;
  int sent; /* flag to indicate if this argument has
               data in flight under non-blocking MPI comms*/
  int opt;  /* flag to indicate if this argument is in use */
} op_arg;
```

在OP2中设置了可以传入多个参数的函数，如下

```
// op_par_loop routine for 12 arguments
//
template <class T0, class T1, class T2, class T3, class T4, class T5, class T6,
          class T7, class T8, class T9, class T10, class T11>
void op_par_loop(void (*kernel)(T0 *, T1 *, T2 *, T3 *, T4 *, T5 *, T6 *, T7 *,
                                T8 *, T9 *, T10 *, T11 *),
                 char const *name, op_set set, op_arg arg0, op_arg arg1,
                 op_arg arg2, op_arg arg3, op_arg arg4, op_arg arg5,
                 op_arg arg6, op_arg arg7, op_arg arg8, op_arg arg9,
                 op_arg arg10, op_arg arg11) {
...
}
```

op_kernel

对核函数的性能统计数据, 如统计时间，性能等

```
typedef struct {
  char const *name; /* name of kernel function */
  int count;        /* number of times called */
  float time;       /* total execution time */
  float plan_time;  /* time spent in op_plan_get */
  float transfer;   /* bytes of data transfer (used) */
  float transfer2;  /* bytes of data transfer (total) */
  float mpi_time;   /* time spent in MPI calls */
} op_kernel;
```

Mesh-based Application (Airfoil)

Airfoil is a non-linear 2D inviscid airfoil code that uses an unstructured grid. It is a finite volume application that solves the 2D Euler equations using a scalar numerical dissipation.

Airfoil is a non-linear 2D inviscid airfoil code that uses an unstructured grid. It is a finite volume application that solves the 2D Euler equations using a scalar numerical dissipation. The algorithm iterates towards the steady state solution, in each iteration using a control volume approach - for example the rate at which the mass changes within a control volume is equal to the net flux of mass into the control volume across the four faces around the cell. This is representative of the 3D viscous flow calculations OP2 supports for production-grade CFD applications (such as the Hydra CFD code at Rolls Royce plc.). Airfoil consists of five parallel loops: save soln, adt calc, res calc, bres calc and update. Out of these, save soln and update are direct loops while the other three are indirect loops. The standard mesh size solved with Airfoil consists of 1.5M edges. In such a mesh the most compute intensive loop, res calc, is called 2000 times during the total execution of the application and performs about 100 floating-point operations per mesh edge. Extensive performance analysis of Airfoil and optimisations have been detailed in our published work . What follows is a step-by-step treatment of the stages involved in developing Airfoil.

翼型是一个非线性二维无粘翼型代码，使用非结构化网格。这是一个有限体积的应用，解决了二维欧拉方程使用标量数值耗散。该算法迭代到稳态解，在每次迭代中使用控制体积方法——例如，控制体积内质量变化的速率等于在单元格四周四个面上进入控制体积的质量净通量。这是生产级CFD应用(如劳斯莱斯plc的Hydra CFD代码)的三维粘性流计算OP2支持的代表。翼型由五个并行循环组成:保存soln, adt calc, res calc, bres calc和update。其中，save soln和update是直接循环，而其他三个是间接循环。用翼型求解的标准网格尺寸由1.5M边组成。在这样的网格中，计算最密集的循环res calc在应用程序的总执行过程中被调用2000次，每个网格边缘执行大约100个浮点运算。广泛的性能分析翼型和优化已详细在我们出版的工作。

使用OP2建模过程如下，期间利用OP2进行了并行算法优化.

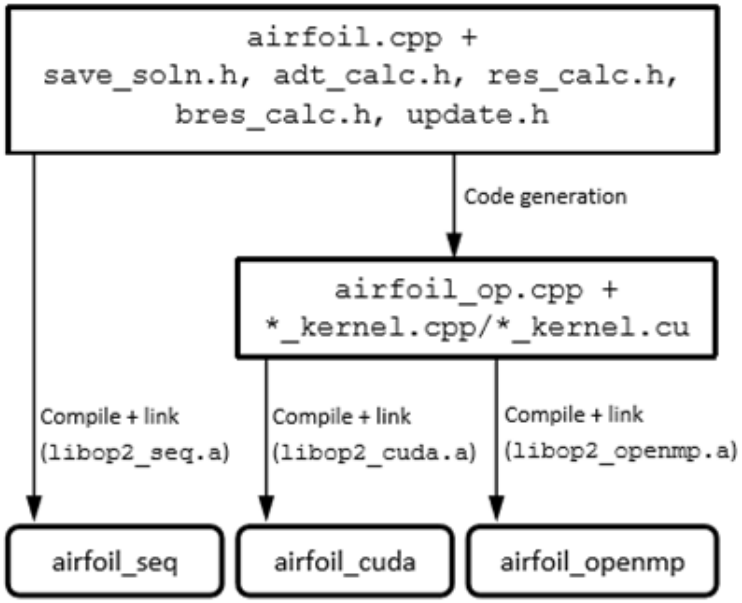


Figure 8: Single-node Code generation and build for Airfoil (with user I/O)

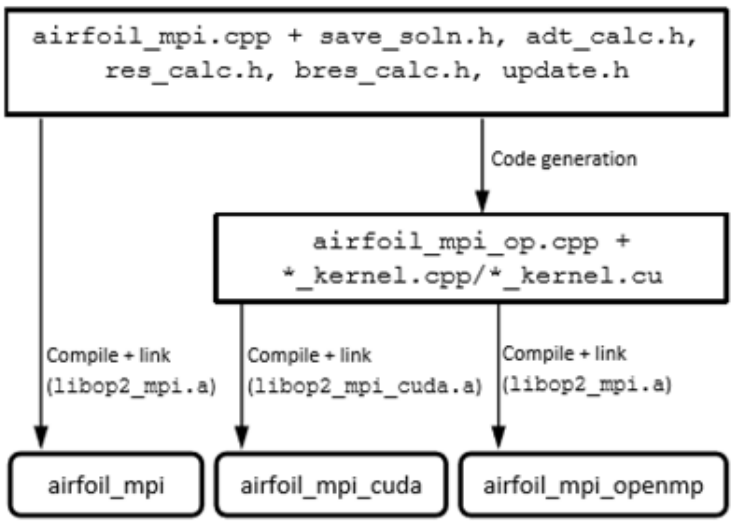


Figure 12: Distributed memory code generation and build for Airfoil (with user I/O)

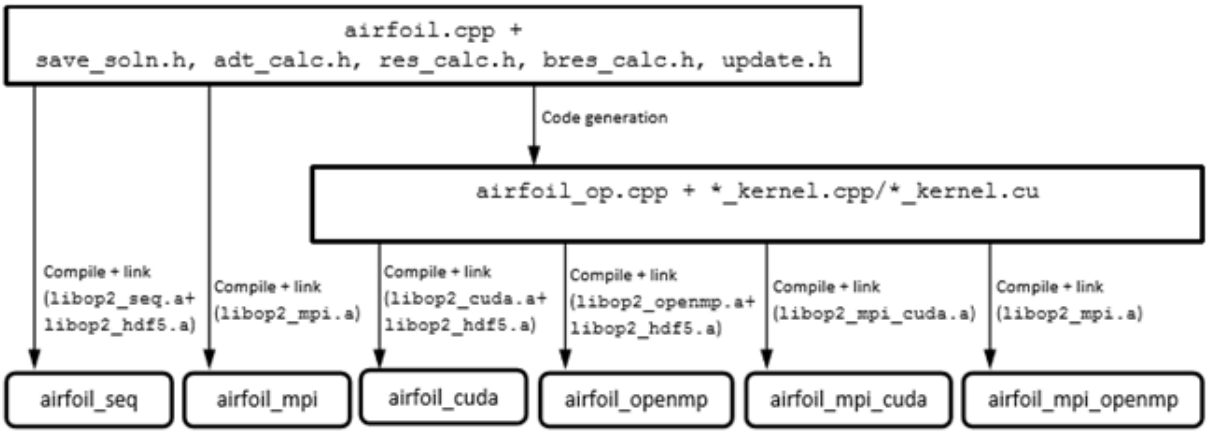


Figure 13: Code generation and build for Airfoil (with OP2 HDF5 file I/O)

Airfoil的主程序

```
int main(int argc, char **argv) {
    // 首先通过这个函数将输入的 mesh 文件地址从程序参数中读出
    // 并且解析输入的 .dat 文件，读入要计算的数据
    op_init(argc, argv, 2);
    // ...
    // 声明了四种类型的数据，分别是节点、边、边界(boundary edge)、网格
    op_set nodes = op_decl_set(nnode, "nodes");
    op_set edges = op_decl_set(nedge, "edges");
    op_set bedges = op_decl_set(nbedge, "bedges");
    op_set cells = op_decl_set(ncell, "cells");

    // 为了知道集合之间的关系，我们还在输入数据中定义了这些集合的映射关系
// 显然一条边有两个端点，因此建立 edge->2 nodes 的映射关系
    op_map pedge = op_decl_map(edges, nodes, 2, edge, "pedge");
    // 显然一条边被两个相邻的网格共用，因此建立 edge->2 cells 的映射关系
    op_map pecell = op_decl_map(edges, cells, 2, ecell, "pecell");
    // 显然一条边缘也有两个端点，因此建立 bedge->2 nodes 的映射关系
    op_map pbedge = op_decl_map(bedges, nodes, 2, bedge, "pbedge");
    // 显然边界只会是一个网格的边，因此建立 bedge->1 cell 的映射关系
    op_map pbecell = op_decl_map(bedges, cells, 1, becell, "pbecell");
    // 由于二维平面的网格有四个顶点，因此建立 cell->4 nodes 的映射关系
    op_map pcell = op_decl_map(cells, nodes, 4, cell, "pcell");

    // 还定义了这些集合的一些数据属性
    op_dat p_bound = op_decl_dat(bedges, 1, "int", bound, "p_bound");
    op_dat p_x = op_decl_dat(nodes, 2, "double", x, "p_x");
    // 用于存放 flow solution
    op_dat p_q = op_decl_dat(cells, 4, "double", q, "p_q");
    // 用于暂存 flow solution 的旧值
    op_dat p_qold = op_decl_dat(cells, 4, "double", qold, "p_qold");
    // 用于存放计算好的 area/timestep
    op_dat p_adt = op_decl_dat(cells, 1, "double", adt, "p_adt");
    // 用于存放 flux residual
    op_dat p_res = op_decl_dat(cells, 4, "double", res, "p_res");
```

```
35
36      for (int iter = 1; iter <= niter; iter++) {
37
38      // save old flow solution
39
40      op_par_loop_save_soln("save_soln",cells,
41                  op_arg_dat(p_q,-1,OP_ID,4,"float",OP_READ),
42                  op_arg_dat(p_qold,-1,OP_ID,4,"float",OP_WRITE));
43
44      // predictor/corrector update loop
45
46      for (int k = 0; k < 2; k++) {
47
48        // calculate area/timstep
49
50        op_par_loop_adt_calc("adt_calc",cells,
51                    op_arg_dat(p_x,0,pcell,2,"float",OP_READ),
52                    op_arg_dat(p_x,1,pcell,2,"float",OP_READ),
53                    op_arg_dat(p_x,2,pcell,2,"float",OP_READ),
54                    op_arg_dat(p_x,3,pcell,2,"float",OP_READ),
55                    op_arg_dat(p_q,-1,OP_ID,4,"float",OP_READ),
56                    op_arg_dat(p_adt,-1,OP_ID,1,"float",OP_WRITE));
57
58        // calculate flux residual
59
60        op_par_loop_res_calc("res_calc",edges,
61                    op_arg_dat(p_x,0,pedge,2,"float",OP_READ),
62                    op_arg_dat(p_x,1,pedge,2,"float",OP_READ),
63                    op_arg_dat(p_q,0,pecell,4,"float",OP_READ),
64                    op_arg_dat(p_q,1,pecell,4,"float",OP_READ),
65                    op_arg_dat(p_adt,0,pecell,1,"float",OP_READ),
66                    op_arg_dat(p_adt,1,pecell,1,"float",OP_READ),
67                    op_arg_dat(p_res,0,pecell,4,"float",OP_INC),
68                    op_arg_dat(p_res,1,pecell,4,"float",OP_INC));
69
70        op_par_loop_bres_calc("bres_calc",bedges,
71                    op_arg_dat(p_x,0,pbedge,2,"float",OP_READ),
72                    op_arg_dat(p_x,1,pbedge,2,"float",OP_READ),
73                    op_arg_dat(p_q,0,pbecell,4,"float",OP_READ),
74                    op_arg_dat(p_adt,0,pbecell,1,"float",OP_READ),
75                    op_arg_dat(p_res,0,pbecell,4,"float",OP_INC),
76                    op_arg_dat(p_bound,-1,OP_ID,1,"int",OP_READ));
77
78        // update flow field
79       //   ...
80      }
```

其中调用的`op_arg_dat`相对于`op_arg`的构造函数

```
1   op_arg op_arg_dat(op_dat dat, int idx, op_map map, int dim, char const *type,
    op_access acc)
```

define mesh in code directly

```
// declare sets, pointers, datasets and global constants

op_set nodes = op_decl_set(nnode, "nodes");
op_set edges = op_decl_set(nedge, "edges");
op_set bedges = op_decl_set(nbedge, "bedges");
op_set cells = op_decl_set(ncell, "cells");

op_map pedge = op_decl_map(edges, nodes, 2, edge, "pedge");
op_map pecell = op_decl_map(edges, cells, 2, ecell, "pecell");
op_map pbedge = op_decl_map(bedges, nodes, 2, bedge, "pbedge");
op_map pbecell = op_decl_map(bedges, cells, 1, becell, "pbecell");
op_map pcell = op_decl_map(cells, nodes, 4, cell, "pcell");

op_dat p_bound = op_decl_dat(bedges, 1, "int", bound, "p_bound");
op_dat p_x = op_decl_dat(nodes, 2, "double", x, "p_x");
op_dat p_q = op_decl_dat(cells, 4, "double", q, "p_q");
// op_dat p_qold  = op_decl_dat(cells ,4,"double",qold ,"p_qold");
// op_dat p_adt   = op_decl_dat(cells ,1,"double",adt  ,"p_adt");
// op_dat p_res   = op_decl_dat(cells ,4,"double",res  ,"p_res");

// p_res, p_adt and p_qold  now declared as a temp op_dats during
// the execution of the time-marching loop

op_decl_const(1, "double", &gam);
op_decl_const(1, "double", &gm1);
op_decl_const(1, "double", &cfl);
op_decl_const(1, "double", &eps);
op_decl_const(1, "double", &mach);
op_decl_const(1, "double", &alpha);
op_decl_const(4, "double", qinf);
```

在执行循环之前，我们要定义如下一些常量

```
// set constants and initialise flow field and residual

op_printf("initialising flow field \n");

gam = 1.4f;
gm1 = gam - 1.0f;
cfl = 0.9f;
eps = 0.05f;

double mach = 0.4f;
double alpha = 3.0f * atan(1.0f) / 45.0f;
double p = 1.0f;
double r = 1.0f;
double u = sqrt(gam * p / r) * mach;
double e = p / (r * gm1) + 0.5f * u * u;

qinf[0] = r;
```

```
  qinf[1] = r * u;
  qinf[2] = 0.0f;
  qinf[3] = r * e;

  for (int n = 0; n < ncell; n++) {
    for (int m = 0; m < 4; m++) {
      q[4 * n + m] = qinf[m];
      res[4 * n + m] = 0.0f;
    }
  }
```

kernel function

在这个例子中，我们使用的kernel function 如下

```
inline void adt_calc(const double *x1, const double *x2, const double *x3,
                     const double *x4, const double *q, double *adt) {
  double dx, dy, ri, u, v, c;

  ri = 1.0f / q[0];
  u = ri * q[1];
  v = ri * q[2];
  c = sqrt(gam * gm1 * (ri * q[3] - 0.5f * (u * u + v * v)));

  dx = x2[0] - x1[0];
  dy = x2[1] - x1[1];
  *adt = fabs(u * dy - v * dx) + c * sqrt(dx * dx + dy * dy);

  dx = x3[0] - x2[0];
  dy = x3[1] - x2[1];
  *adt += fabs(u * dy - v * dx) + c * sqrt(dx * dx + dy * dy);

  dx = x4[0] - x3[0];
  dy = x4[1] - x3[1];
  *adt += fabs(u * dy - v * dx) + c * sqrt(dx * dx + dy * dy);

  dx = x1[0] - x4[0];
  dy = x1[1] - x4[1];
  *adt += fabs(u * dy - v * dx) + c * sqrt(dx * dx + dy * dy);

  *adt = (*adt) / cfl;
}
```

Performance report

而在这个案例中，performance report 如下

```
// initialise timers for total execution wall time
op_timers(&cpu_t1, &wall_t1);

// main time-marching loop
op_timers(&cpu_t2, &wall_t2);
op_timing_output();
op_printf("Max total runtime = %f\n", wall_t2 - wall_t1);
```

在开始前后使用计时器进行计时，来得知performance

而执行的历史纪录有用户调用查看

```
    // print iteration history
    rms = sqrt(rms / (double)g_ncell);
    if (iter % 100 == 0)
      op_printf(" %d  %10.5e \n", iter, rms);

    if (iter % 1000 == 0 &&
        g_ncell == 720000) { // defailt mesh -- for validation testing
      // op_printf(" %d  %3.16f \n",iter,rms);
      double diff = fabs((100.0 * (rms / 0.0001060114637578)) - 100.0);
      op_printf("\n\nTest problem with %d cells is within %3.15E %% of the "
                "expected solution\n",
                720000, diff);
      if (diff < 0.00001) {
        op_printf("This test is considered PASSED\n");
      } else {
        op_printf("This test is considered FAILED\n");
      }
    }
```

MPI 并行加速

在airfoil_mpi.cpp中有相对于的体现

```
int main(int argc, char **argv) {
  // OP initialisation
  op_init(argc, argv, 2);

  // MPI for user I/O
  int my_rank;
  int comm_size;
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
  MPI_Comm_size(MPI_COMM_WORLD, &comm_size);

```

在计算时也调用了相对于的MPI并行框架的函数

```
 MPI_Scatterv(g_array, sendcnts, displs, MPI_DOUBLE, l_array,
              l_size * elem_size, MPI_DOUBLE, MPI_ROOT, MPI_COMM_WORLD);
```

Domain Model

抽象

OP2 Abstraction

sets (e.g. nodes, edges, faces) datasets (e.g. flow variables) pointers (e.g. from edges to nodes) parallel loops operate over all members of one set datasets have at most one level of indirection user specifies how data is used (e.g. read-only, write-only, increment)

Grid

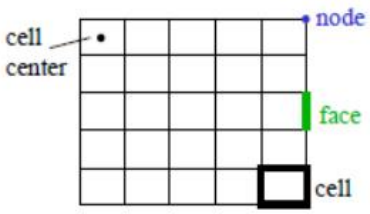
网格划分可以分为三大类，结构化、非结构化和混合型（结构和非结构的复合）。在结构网格中，遵循一个重复的样式，每个节点按指定算法唯一定义。而在非结构网格中，通常所有节点随机定义，没有规则的样式可循。结构网格中，在空间内，一个节点的周围有相同数量的节点，而对于非结构网格则没有这样的要求。

• The grid: – Designates the cells or elements on which the flow is solved. – Is a discrete representation of the geometry of the problem. – Has cells grouped into boundary zones where b.c.'s are applied. • The grid has a significant impact on: – Rate of convergence (or even lack of convergence). – Solution accuracy. – CPU time required. • Importance of mesh quality for good solutions. – Grid density. – Adjacent cell length/volume ratios. – Skewness. – Tet vs. hex. – Boundary layer mesh. – Mesh refinement through adaption.

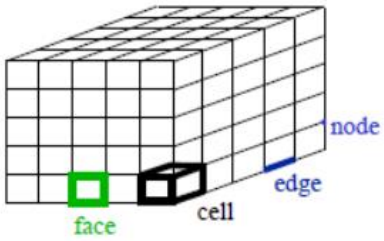
Terminology in Grid

Terminology

- Cell = control volume into which domain is broken up.
- Node = grid point.
- Cell center = center of a cell.
- Edge = boundary of a face.
- Face = boundary of a cell.
- Zone = grouping of nodes, faces, and cells:
  - Wall boundary zone.
  - Fluid cell zone.
- Domain = group of node, face and cell zones.



2D computational grid



3D computational grid

7

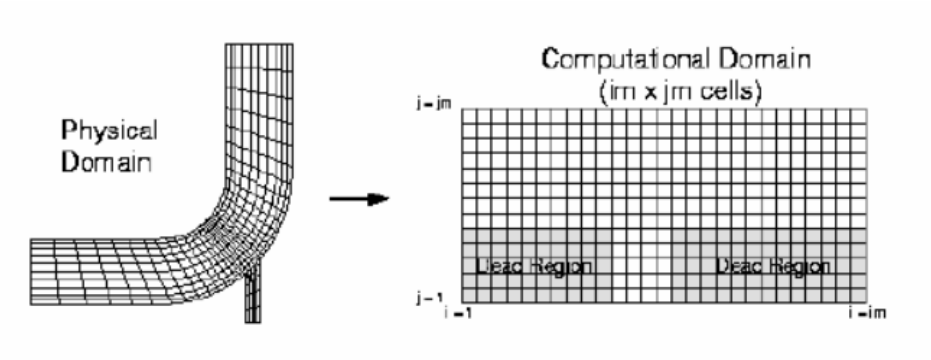
Structured Grid

A regular grid is a tessellation of n-dimensional Euclidean space by congruent parallelotopes (e.g. bricks). Grids of this type appear on graph paper and may be used in finite element analysis, finite volume methods, finite difference methods, and in general for discretization of parameter spaces. Since the derivatives of field variables can be conveniently expressed as finite differences, structured grids mainly appear in finite difference methods. Unstructured grids offer more flexibility than structured grids and hence are very useful in finite element and finite volume methods.

规则网格是由相等的平行色块(如砖块)构成的 n 维欧几里得空间的镶嵌。这种类型的网格出现在图形纸上，可用于有限元分析、有限体积法、有限差分法，一般用于参数空间的离散化。由于场变量的导数可以方便地表示为有限差分，结构网格主要出现在有限差分法中。非结构化网格比结构化网格具有更大的灵活性，因此在有限元和有限体积方法中非常有用。

Grid types: structured grid

- Single-block, structured grid.
  - i,j,k indexing to locate neighboring cells.
  - Grid lines must pass all through domain.
- Obviously can't be used for very complicated geometries.



Unstructured Grid

An unstructured (or irregular) grid is a tessellation of a part of the Euclidean plane or Euclidean space by simple shapes, such as triangles or tetrahedra, in an irregular pattern. Grids of this type may be used in finite element analysis when the input to be analyzed has an irregular shape.

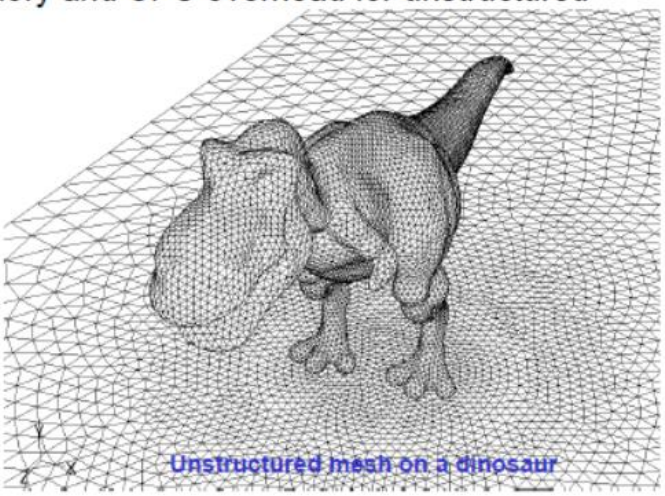
Unlike structured grids, unstructured grids require a list of the connectivity which specifies the way a given set of vertices make up individual elements (see graph (data structure)).

非结构化(或不规则)网格是欧几里得平面或欧几里得空间的一部分通过简单形状(如三角形或四面体)以不规则模式镶嵌而成的网格。当需要分析的输入具有不规则形状时，这种类型的网格可用于有限元分析。

与结构化网格不同，非结构化网格需要一个连接性列表，该列表指定了给定顶点集组成单个元素的方式(参见图的数据结构)。

Grid types: unstructured

- Unstructured grid.
  - The cells are arranged in an arbitrary fashion.
  - No i,j,k grid index, no constraints on cell layout.
- There is some memory and CPU overhead for unstructured referencing.



12

Comparison between Structured Grid and Unstructured Grid

网格划分是利用 CFD 解决问题过程中重要的一环。正如 Pointwise 在 2013 年的文章中所指出，如果认为，对于今日广泛使用的非结构或混合网格，结构网格很大程度上是多余的，这种看法也可以理解。

控制热流体现象的偏微分方程常常不具有解析解，除了非常简单的某些案例。所以，为了分析流动，将几何空间分割成有限元素或体积，对其采用一阶或二阶格式进行离散。生成的元素是二维的三角形或四边形，三维的四面体或六面体。**合适的网格对于精确求解、快速收敛以及减少数值耗散是必要的**。

从严格意义上讲,结构化网格是指网格区域内所有的内部点都具有相同的毗邻单元.结构化网格生成技术有大量的文献资料.结构化网格有很多优点: 1.它可以很容易地实现区域的边界拟合,适于流体和表面应力集中等方面的计算. 2.网格生成的速度快. 3.网格生成的质量好 4.数据结构简单 5.对曲面或空间的拟合大多数采用参数化或样条插值的方法得到,区域光滑,与实际的模型更容易接近. 它的最典型的缺点是适用的范围比较窄.尤其随着近几年的计算机和数值方法的快速发展,人们对求解区域的复杂性的要求越来越高,在这种情况下,结构化网格生成技术就显得力不从心了.

非结构化网格技术的分类,可以根据应用的领域分为应用于差分法的网格生成技术（常常成为 grid generation technology）和应用于有限元方法中的网格生成技术（常常成为 mesh generation technology）,应用于差分计算领域的网格要除了要满

足区域的几何形状要求以外,还要满足某些特殊的性质（如垂直正交,与流线平行正交等）,因而从技术实现上来说就更困难一些.基于有限元方法的网格生成技术相对非常自由,对生成的网格只要满足一些形状上的要求就可以了.

Mesh file as a text file

参考：http://www.cnblogs.com/Adellbengbeng/p/4209369.html

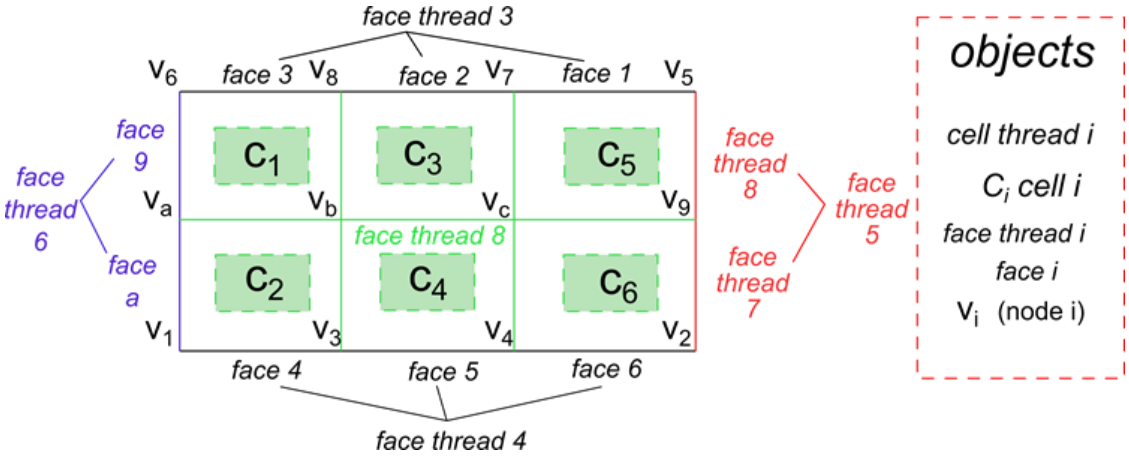
将 Mesh 结构以文本的形式存储在磁盘上

存储方式有多种，主要是与其在的数据结构有关

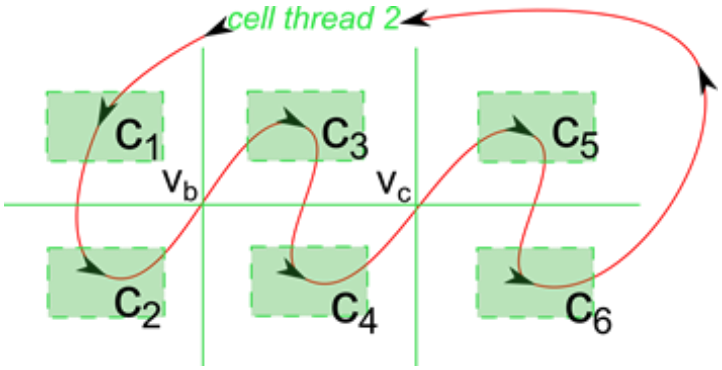
一种存储方式如下所示

0. 关于 nodes,faces,cells 的说明

从底层到顶层的组成顺序看，nodes 组成 faces，faces 围成 cells，且为便于遍历 faces 和 cells，Fluent 引入了 face thread 和 cell thread，可将 thread 看成链接 face 或 cell 指针的链表结构



节点，面及单元格



单元格线程

存储形式

1. dimensions（网格维度）

标号：2

格式：

```
(2 ND)
```

其中N为网格的维度，可为2或3

实例：

```
(0 "Dimension:")
(2 2)
```

2. nodes（节点数据）

标号：10

格式：

```
(10 (zone-id first-index last-index type ND)(x1
y1 z1  x2 y2 z2... ))
```

*如果zone-id=0, first-index将是1，last-index等于节点数，type设置为1, ND是网格维度，后面不跟坐标数据。此时相当于对nodes的整体说明

*如果zone-id大于0，表示结构体中的nodes属于编号zone-id的zone区域。此时first-index和last-index为该zone区域的节点编号，type设置为1，ND为可选参数，表示网格维度。当ND=2时，节点数据不显示z坐标

实例：

```
(10 (0 1 C 1 2))
(10 (1 1 C 1 2)(
   0.0000000000e+000    0.0000000000e+000
   2.0000000000e+000    0.0000000000e+000
   6.6666666667e-001    0.0000000000e+000
   1.3333333333e+000    0.0000000000e+000
   2.0000000000e+000    1.0000000000e+000
   0.0000000000e+000    1.0000000000e+000
   1.3333333333e+000    1.0000000000e+000
   6.6666666667e-001    1.0000000000e+000
   2.0000000000e+000    5.0000000000e-001
   0.0000000000e+000    5.0000000000e-001
   6.6666666667e-001    5.0000000000e-001
   1.3333333333e+000    5.0000000000e-001
))
```

3. faces（面及其线程)

标号：13

格式：

```
(13 (zone-id first-index last-index bc-type
face-type))
```

*zone-id=0时，语句说明面的数量(last-index – first-index + 1)，且不写出bc-type

*zone-id大于0时，为面线程编号，firt-index和last-index分别为线程中面标号的边界值

*bc-type:

bc-type	*decirption*
2	interior
3	wall
4	pressure-inlet,inlet-vent,intake-fan
5	pressure-outlet,exhaust-fan,outlet-vent
7	symmetry
8	periodic-shadow
9	pressure-far-field
10	velocity-inlet
12	periodic
14	fan,porous-jump,radiator
20	mass-flow-inlet
24	interface
31	parent(hanging node)
36	outflow
37	axis

其他数据，每一行表示一个face:

```
n0 n1 n2 c0 c1
```

n*表示节点编号，对于2维网格，n2不显示；c*表示face的邻近cell编号，c0按右手法则确定，c1在face的另一边，在边界处c0或c1为0。

*当网格为混合类型时，即face-type=0，每一行说明面的语句应以节点数目开头：

```
x n0 n1 ... nf c0 c1
```

x表示面上的节点数，nf表示最后一个节点。

4. cells

标号：12

格式：

```
(12 (zone-id first-index last-index type
element-type))
```

*zone-id=0时，语句用于说明cell的数目，若last-index=0则表示文件中无cell。type=0，element-type不显示

*zone-id大于0时，表示单元格线程，

type = 4 for hex
type = 2 for tet
type = 5 for pyramid

实例：

```
(0 "Faces:")
(13(0 1 11 0))
(13(3 1 3  3 0)(
2 5 7 5 0
2 7 8 3 0
2 8 6 1 0
))
(13(4 4 6  3 0)(
2 1 3 2 0
2 3 4 4 0
2 4 2 6 0
))
```

5. zones

标号：45

格式：

```
(45 (id type name)())
```

id为区域编号

type为类型

name为区域名称

实例：

```
(0 "Zones:")
(45 (2 fluid fluid)())
(45 (3 wall up)())
(45 (4 wall down)())
(45 (5 pressure-outlet outlet)())
(45 (6 velocity-inlet inlet)())
(45 (8 interior default-interior)())
```

完成后的 msh 文件大致如下

```
(0 "GAMBIT to Fluent File")

(0 "Dimension:")
(2 2)

(10 (0 1 C 1 2))
(10 (1 1 C 1 2)(
   0.0000000000e+000    0.0000000000e+000
   2.0000000000e+000    0.0000000000e+000
   6.6666666667e-001    0.0000000000e+000
   1.3333333333e+000    0.0000000000e+000
   2.0000000000e+000    1.0000000000e+000
```

```
   0.0000000000e+000    1.0000000000e+000
   1.3333333333e+000    1.0000000000e+000
   6.6666666667e-001    1.0000000000e+000
   2.0000000000e+000    5.0000000000e-001
   0.0000000000e+000    5.0000000000e-001
   6.6666666667e-001    5.0000000000e-001
   1.3333333333e+000    5.0000000000e-001
))

(0 "Faces:")
(13(0 1 11 0))
(13(3 1 3  3 0)(
2 5 7 5 0
2 7 8 3 0
2 8 6 1 0
))
(13(4 4 6  3 0)(
2 1 3 2 0
2 3 4 4 0
2 4 2 6 0
))
(13(5 7 8  5 0)(
2 2 9 6 0
2 9 5 5 0
))
(13(6 9 a  a 0)(
2 6 a 1 0
2 a 1 2 0
))
(13(8 b 11 2 0)(
2 a b 1 2
2 b 8 1 3
2 3 b 2 4
2 b c 3 4
2 c 7 3 5
2 4 c 4 6
2 c 9 5 6
))

(0 "Cells:")
(12 (0 1 6 0))
(12 (2 1 6 1 3))

(0 "Zones:")
(45 (2 fluid fluid)())
(45 (3 wall up)())
(45 (4 wall down)())
(45 (5 pressure-outlet outlet)())
(45 (6 velocity-inlet inlet)())
(45 (8 interior default-interior)())
```

还有一种是以 XML 形式存储

```
<library_geometries>
    <geometry id="Tall_bodyShape" name="Tall_bodyShape">
        <mesh>
            <source id="Tall_bodyShape-positions" name="position">
                <float_array id="Tall_bodyShape-positions-array" count=
"2910"> *** lots of numbers *** </float_array>
                <technique_common>
                    <accessor source="#Tall_bodyShape-positions-array"
count="970" stride="3">
                        <param name="X" type="float"/>
                        <param name="Y" type="float"/>
                        <param name="Z" type="float"/>
                    </accessor>
                </technique_common>
            </source>
            <source id="Tall_bodyShape-normals" name="normal">
                <float_array id="Tall_bodyShape-normals-array" count="3
948"> *** lots of numbers *** </float_array>
                <technique_common>
                    <accessor source="#Tall_bodyShape-normals-array" co
unt="1316" stride="3">
                        <param name="X" type="float"/>
                        <param name="Y" type="float"/>
                        <param name="Z" type="float"/>
                    </accessor>
                </technique_common>
            </source>
            <source id="Tall_bodyShape-UVChannel_1" name="UVChannel_1">
                <float_array id="Tall_bodyShape-UVChannel_1-array" coun
t="2892"> *** lots of numbers *** </float_array>
                <technique_common>
                    <accessor source="#Tall_bodyShape-UVChannel_1-array
" count="1446" stride="2">
                        <param name="S" type="float"/>
                        <param name="T" type="float"/>
                    </accessor>
                </technique_common>
            </source>
            <vertices id="Tall_bodyShape-vertices">
                <input semantic="POSITION" source="#Tall_bodyShape-posi
tions"/>
            </vertices>
            <triangles material="Tall_bodySG" count="1883">
                <input semantic="VERTEX" source="#Tall_bodyShape-vertic
es" offset="0"/>
                <input semantic="NORMAL" source="#Tall_bodyShape-normal
s" offset="1"/>
                <input semantic="TEXCOORD" source="#Tall_bodyShape-UVCh
```

```
annel_1" offset="2" set="0"/>
                <p> *** lots of numbers *** </p>
            </triangles>
        </mesh>
        <extra>
            <technique profile="MAYA">
                <double_sided>1</double_sided>
            </technique>
        </extra>
    </geometry>
</library_geometries>
```

Data Structure in Mesh

Structured Mesh

Each cell in the grid can be addressed by index (i, j) in two dimensions or (i, j, k) in three dimensions, and each vertex has coordinates

in 2D or

 in 3D for some real numbers *dx*, *dy*, and *dz* representing the grid spacing.

Unstructured Mesh

Unlike structured grids, unstructured grids require a list of the connectivity which specifies the way a given set of vertices make up individual elements (see graph (data structure)).

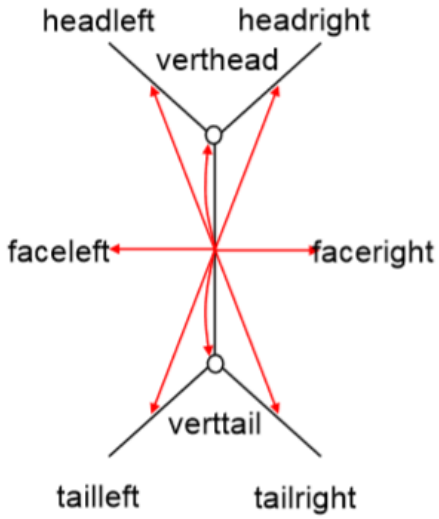
图的存储

A graph data structure consists of a finite (and possibly mutable) set of *vertices* or *nodes* or *points*, together with a set of unordered pairs of these vertices for an undirected graph or a set of ordered pairs for a directed graph. These pairs are known as *edges*, *arcs*, or *lines* for an undirected graph and as *arrows*, *directed edges*, *directed arcs*, or *directed lines* for a directed graph. The vertices may be part of the graph structure, or may be external entities represented by integer indices or references.

A graph data structure may also associate to each edge some *edge value*, such as a symbolic label or a numeric attribute (cost, capacity, length, etc.).

Winged-edge

```
struct Edge {
   Edge *headleft,*headright,
*tailleft,*tailright;
   Face *faceleft,*faceright;
   Vertex *verthead, *verttail;
   // edge data
} ;
struct Face {
  Edge* edge;
  // face data
};
struct Vertex {
  Edge* edge;
 // vertex data
};
```

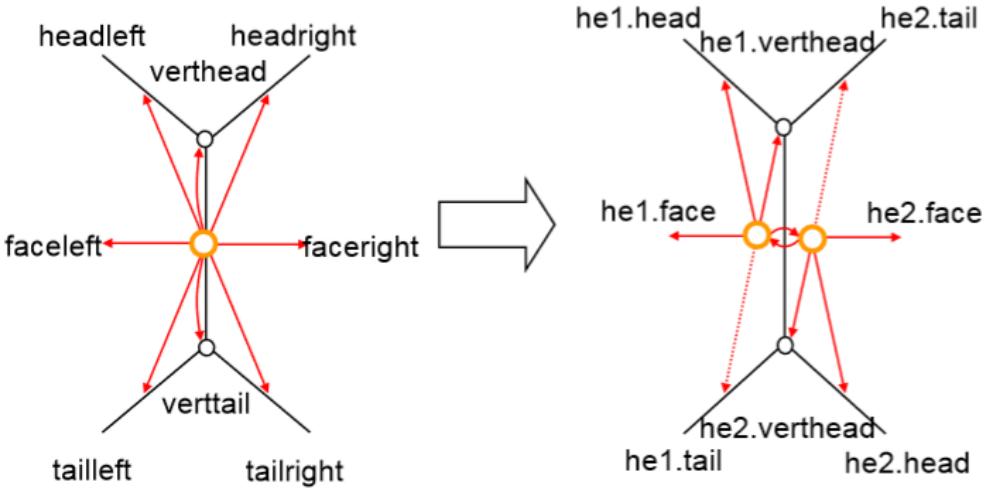


red arrows indicate pointers

Half-edge

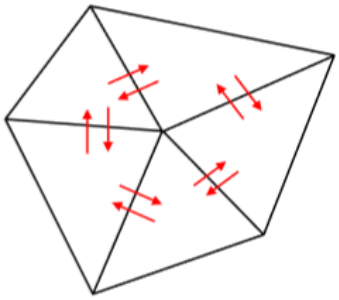
Split each winged-edge data structure into 2;

- advantage: FE, VE traversals do not require "ifs" in code, consistent orientation



Face-based data structure

Primarily for triangle meshes



```
Face {
  Face*nbr[3];
  Vertex* vert[3];
}
```

6 pointers per triangle

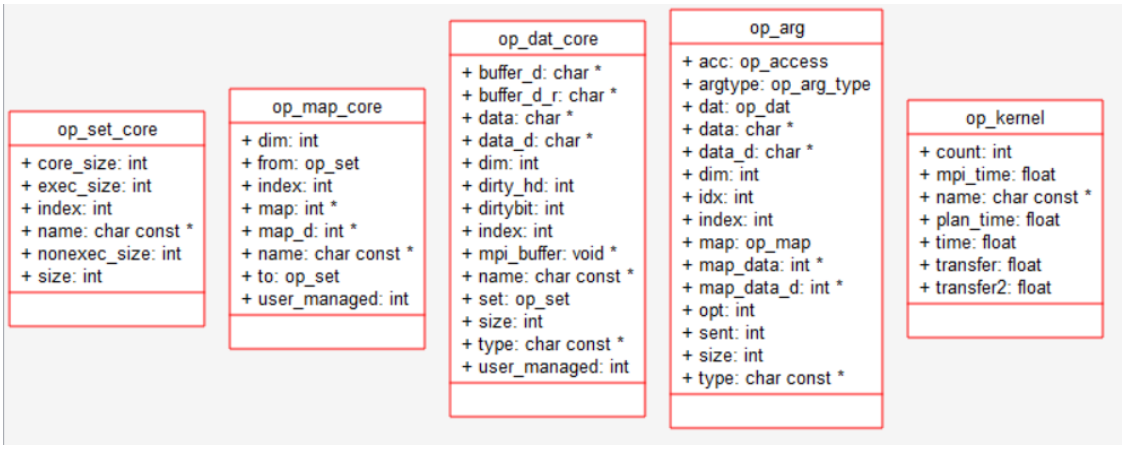
1 per vertex, no edge records

(3/2*6+1)*N = 10*N vs. 27N for winged-edge

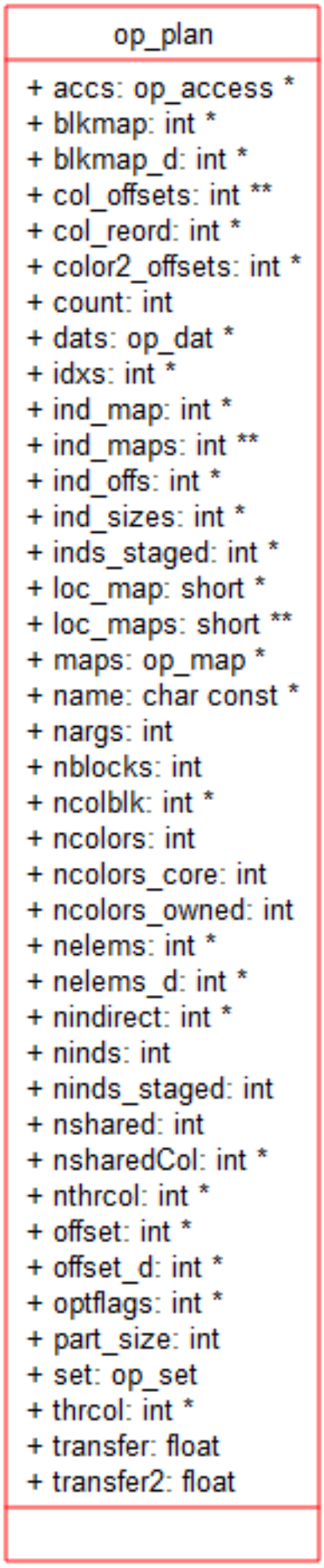
在 OP2 中还有以下结构存储（见于下一部分）

Mesh structure in memory

op*lib*core.h 中关键成员的 UML 视图如下



在 op*rt*support.h 中 op_plan 的 UML 视图如下所示


op_plan
+ accs: op_access *
+ blkmap: int *
+ blkmap_d: int *
+ col_offsets: int **
+ col_reord: int *
+ color2_offsets: int *
+ count: int
+ dats: op_dat *
+ idxs: int *
+ ind_map: int *
+ ind_maps: int **
+ ind_offs: int *
+ ind_sizes: int *
+ inds_staged: int *
+ loc_map: short *
+ loc_maps: short **
+ maps: op_map *
+ name: char const *
+ nargs: int
+ nblocks: int
+ ncolblk: int *
+ ncolors: int
+ ncolors_core: int
+ ncolors_owned: int
+ nelems: int *
+ nelems_d: int *
+ nindirect: int *
+ ninds: int
+ ninds_staged: int
+ nshared: int
+ nsharedCol: int *
+ nthrcol: int *
+ offset: int *
+ offset_d: int *
+ optflags: int *
+ part_size: int
+ set: op_set
+ thrcol: int *
+ transfer: float
+ transfer2: float

op2 can use the distributed memory systems

Unstructured meshes, unlike structured meshes, use connectivity information to specify the mesh topology. The OP2 approach to the solution of unstructured mesh problems (based on ideas developed in its predecessor OPlus involves breaking down the algorithm into four distinct parts: (1) sets, (2) data on sets, (3) connectivity (or mapping) between the sets and (4) operations over sets. This leads to an API through which any mesh or graph can be completely and abstractly de ned. Depending on the application, a set can consist of nodes, edges, triangular faces, quadrilateral faces, or other elements. Associated with these sets are data (e.g. node coordinates, edge weights) and mappings between sets which de ne how elements of one set connect with the elements of another set.

One key implementation choice is how to store datasets in which there are multiple items for each set element. For example, in the “airfoil” testcase there are four flow variables for each cell. The two alternatives are: • SoA: for each component, store the data for all of the set elements as a contiguous block.

• AoS: for each set element, store all of the components together as a small contiguous block;

set

unstructured grids can be described by a number of sets. Depending on the application, these sets might be of nodes, edges, faces, cells of a variety of types, far-field boundary nodes, wall boundary faces, etc.

map

Associated with these are data (e.g. coordinate data at nodes) and mappings to other sets (e.g. edge mapping to the two nodes at each end of the edge). All of the numerically-intensive operations can then be described as a loop over all members of a set, carrying out some operations on data associated directly with the set or with another set through a mapping.

dat

OP dataset ID (store data in dataset). “Datasets” are the data associated with the sets, such as flow variables or edge weights, which are the arguments to the parallel loop functions.

arg

The op arg arguments in op par loop are provided by one of the following routines, one for global constants and reductions, and the other for OP2 datasets. In the future there will be a third one for sparse matrices to support the needs of finite element calculations.

plan

info on plan construction;The first part of the op plan struct stores the input arguments used to construct the plan. These are needed to determine whether the inputs for a new parallel loop match an existing plan.

Mesh-based Applications

CFD

CFD，英语全称(Computational Fluid Dynamics），即计算流体动力学,是流体力学的一个分支，简称 CFD。CFD 是近代流体力学，数值数学和计算机科学结合的产物，是一门具有强大生命力的交叉科学。它以电子计算机为工具，应用各种离散化的数学方法，对流体力学的各类问题进行数值实验、计算机模拟和分析研究，以解决各种实际问题。

CFD 是计算流体力学（Computational Fluid Dynamics）的简称，是流体力学和计算机科学相互融合的一门新兴交叉学科，它从计算方法出发，利用计算机快速的计算能力得到流体控制方程的近似解。CFD 兴起于 20 世纪 60 年代，随着 90 年代后计算机的迅猛发展，CFD 得到了飞速发展，逐渐与实验流体力学一起成为产品开发中的重要手段。

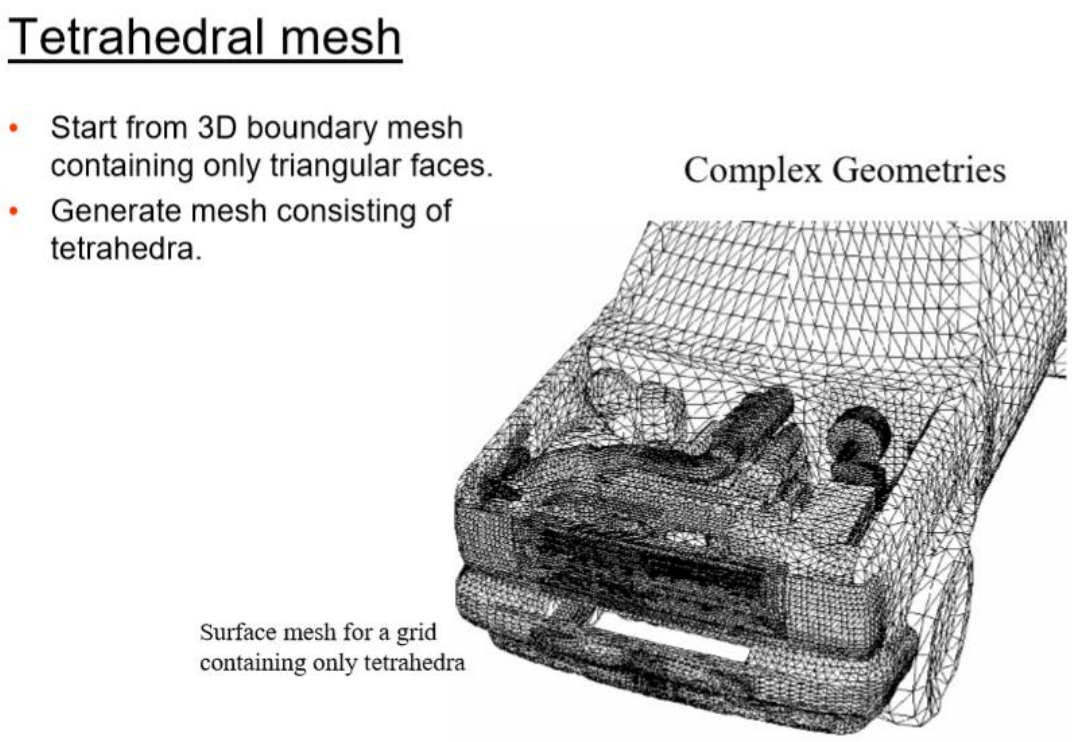
CFD 软件通常指商业化的 CFD 程序，具有良好的人机交互界面，能够使使用者无需精通 CFD 相关理论就能够解决实际问题。

计算流体力学和相关的计算传热学，计算燃烧学的原理是用数值方法求解非线性联立的质量、能量、组分、动量和自定义的标量的微分方程组，求解结果能预报流动、传热、传质、燃烧等过程的细节，并成为过程装置优化和放大定量设计的有力工具。计算流体力学的基本特征是数值模拟和计算机实验，它从基本物理定理出发，在很大程度上替代了耗资巨大的流体动力学实验设备，在科学研究和工程技术中产生巨大的影响。 是目前国际上一个强有力的研究领域, 是进行传热、传质、动量传递及燃烧、多相流和化学反应研究的核心和重要技术, 广泛应用于航天设计、汽车设计、生物医学工业、化工处理工业、涡轮机设计、半导体设计、HVAC&R 等诸多工程领域,板翅式换热器设计是 CFD 技术应用的重要领域之一。

CFD 在最近 20 年中得到飞速的发展, 除了计算机硬件工业的发展给它提供了坚实的物质基础外, 还主要因为无论分析的方法或实验的方法都有较大的限制, 例如由于问题的复杂性, 既无法作分析解, 也因费用昂贵而无力进行实验确定, 而 CFD 的方法正具有成本低和能模拟较复杂或较理想的过程等优点。经过一定考核的 CFD 软件可以拓宽实验研究的范围, 减少成本昂贵的实验工作量。在给定的参数下用计算机对现象进行一次数值模拟相当于进行一次数值实验, 历史上也曾有过首先由 CFD 数值模拟发现新现象而后由实验予以证实的例子。CFD 软件一般都能推出多种优化的物理模型，如定常和非定常流动、层流、紊流、不可压缩和可压缩流动、传热、化学反应等等。对每一种物理问题的流动特点, 都有适合它的数值解法, 用户可对显式或隐式差分格式进行选择, 以期在计算速度、稳定性和精度等方面达到最佳。CFD

软件之间可以方便地进行数值交换, 并采用统一的前、后处理工具, 这就省却了科研工作者在计算机方法、编程、前后处理等方面投入的重复、低效的劳动, 而可以将主要精力和智慧用于物理问题本身的探索上。

如汽车发动机建模



Kernel functions

The OP2 design uses hierarchical parallelism with two principal levels. At the higher level, OP2 is parallelized across distributed-memory clusters using MPI messagepassing. This uses essentially the same implementation approach as the original OPlus. The domain is partitioned among the compute nodes of the cluster, and import/export halos are constructed for message-passing. Data conflicts when incrementing indirectly referenced datasets are avoided by using an “owner-compute” model, in which each process performs the computations which are required to update data owned by that partition. This approach involves some amount of redundant computation; for example, if the computation for a particular edge results in increments to the data at two nodes belonging to different partitions, then both of those partitions will need to perform this edge computation. However, there will be a minimal level of redundant computation if the cluster satisfies the first of our key assumptions:

```
void res(double* edge, double* cell0, double* cell1){
  *cell0 += *edge;
  *cell1 += *edge;
}
op_par_loop(res,"residual_calculation", edges,
   op_arg(dedges, -1, OP_ID,  1, "double", OP_READ),
   op_arg(dcells,  0, pecell, 1, "double", OP_INC),
   op_arg(dcells,  1, pecell, 1, "double", OP_INC));
```

The elemental kernel function takes 3 arguments in this case and the parallel loop declaration requires the access method of each to be declared (OP_INC, OP_READ, etc). OP_ID indicates that the data in `dedges` is to be accessed without any indirection (i.e. directly). `decells` on the other hand is accessed through the `pecell` mapping using the given index (0 and 1). The dimension (or cardinality) of the data (in this example 1, for all data) is also declared.

The OP2 compiler handles the architecture specific code generation and parallelization. An application written using the OP2 API will be parsed through the OP2 compiler and will produce a modified main program and back-end specific code. These are then compiled using a conventional compiler (e.g. gcc, icc, nvcc) and linked against platform specific OP2 back-end libraries to generate the final executable. In the OP2 project we currently have two prototype compilers, one written in MATLAB which only parses OP2 calls and a second source-to-source translator built using the ROSE compiler framework [10] which is capable of full source code analysis. Preliminary details of the ROSE source-to-source translator can be found in [16]. The slightly verbose API was needed as a result of the initial MATLAB prototype parser but also facilitate consistency checks to identify user errors during application development.

Algorithm

常见算法

（1）Octree 算法。

Octree 算法是 ICEM CFD 最重要的生成四面体网格的算法。绝大多数四面体网格是在生成三角形面网格后，基于面网格生成体网格。这种算法的困难在于细长表面、缝隙等模型的局部细节处理难度较大。而 Octree 算法首先生成独立的体网格，然后进行网格调整，将网格映射到面、线和点上。网格与几何模型表面局部细节能按需求处理，可以选择捕捉局部细节，也可以忽略局部细节，如图 2-156 所示。

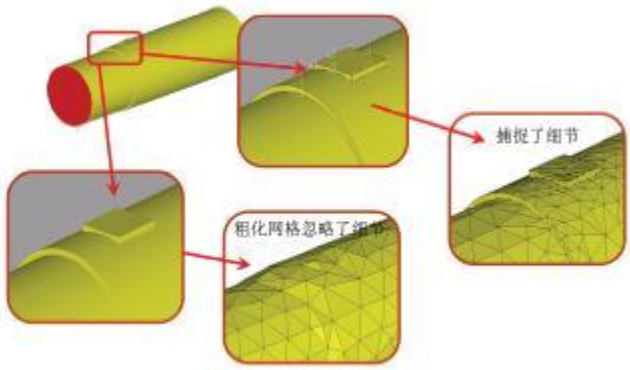


图 2-156　利用 Octree 算法划分网格并捕捉细节

（2）快速 Delaunay 阵面推进算法。

使用这种方法可以得到非常有效的网格，生成的网格没有使用前沿推进方法生成的网格平滑，这种方法可能会出现稳定性的问题，特别是在初始化的阶段，会存在非常扭曲的网格元素，一种折衷的方法是用前沿推进方法生成内部的点，这样可以得到更平滑的网格，可以被用来产生非结构化的网格。如果已经有质量较好的表面网格，那么从表面网格开始（即从八叉树或者从导入的部分面网格）开始划分。

（3）Advanced Front（前沿推进）光顺算法。

对边界进行离散化（如在二维空间里用一组多边形来近似），这就是最初的前沿。在区域内加入三角形或四边形，并且加入的三角形或四边形中至少有一条边与前沿重合。在每一步中需要更新前沿。当不再有新的前沿留下时，网格的生成也就完成了。这种方法要求整个区域的边界是封闭的，但是对于边界不封闭的区域，前沿也可以被推进，直到前沿和区域的距离相等为止。虽然前沿推进方法的实现思想很简单，但实现这一方法的细节很复杂，并且生成网格所花费时间较多且不稳定。

该方法生成网格的速度和阵面推进的一样快，但是使用阵面推进法是从表面向内部推进网格，主要利用 GE/CFX 的算法，其网格尺度变化更加渐进、更精细，表面网格质量必须相当高，才能够自适应检查，并填充小缝隙。图 2-157 所示为以上 3 种网格划分方法划分网格结果的比较。

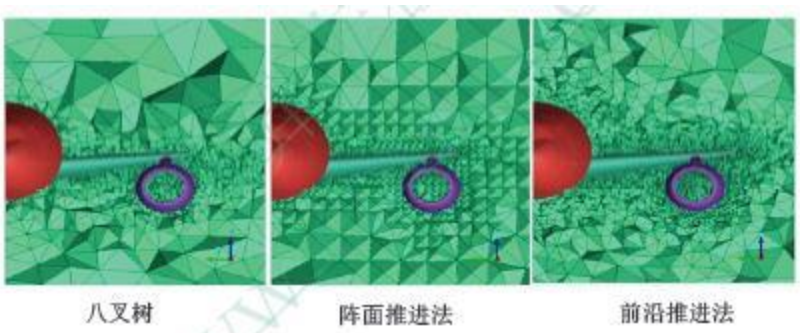


图 2-157　3 种网格划分方法的比较

Delaunay 网格划分算法

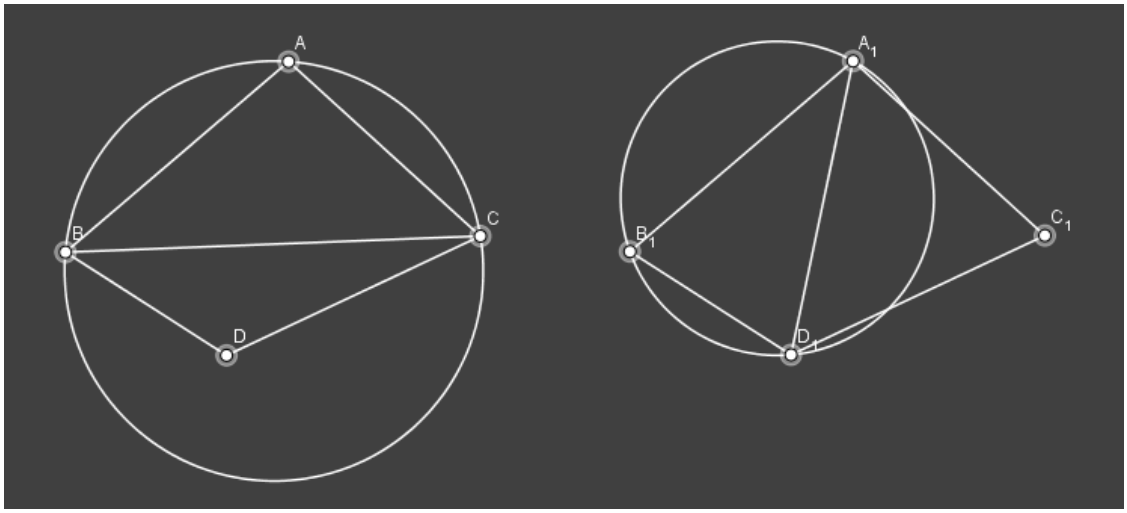
参考：https://seanwangjs.github.io/2017/09/05/delaunay-mesh-generation.html

将一块区域进行网格化是计算几何中的一项重要议题，基于化繁为简的思想，任何复杂的模型都能分解成简单的几何元素。在应用上，整体的非线性行为，便可以由局部的线性拼接得到。

举个简单的例子，一段向上的楼梯，老远看去就是一条斜向上的直线，但实际上是由一段段水平和竖直的线段连成的。从某种意义上讲，将斜线表述成水平线和竖直线的组合降低了描述的层次，但代价是我们需要维护大量的低层次信息。或许对于一段斜直线来讲，这种代价并不值得，但对于更高层次的复杂信息，局部简化是我们目前唯一的办法。

Delaunay 网格划分不是一种具体的算法，而是一种规范，它确保了网格中的三角形的形状处于可控的状态。这是网格划分中不得不考虑的问题，如果同一个单元中的节点离得老远，那么在插值的时候，其误差便没有保证，因为我们知道插值算法的精度依赖于距离。

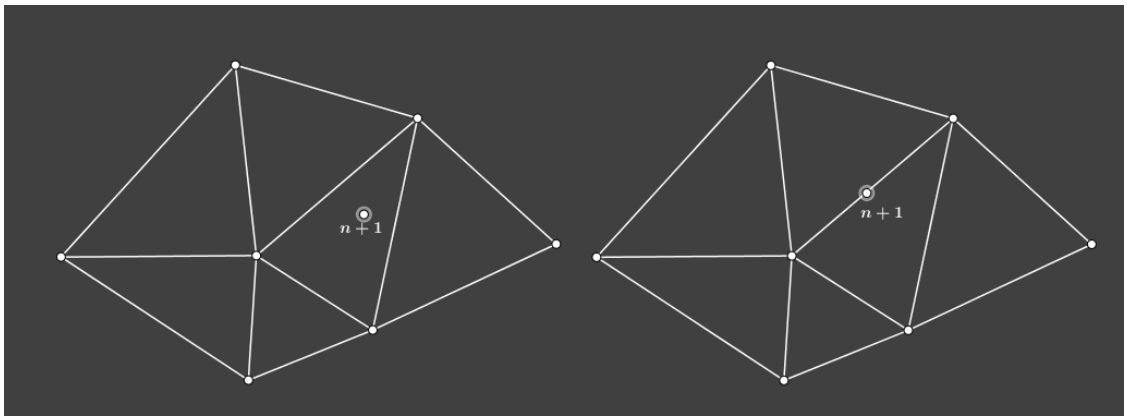
考虑网格中相邻边的两个三角形



按左边连接方式，会发现 D 点位于三角形 ABC 的外接圆内，而如果换成右边的连接，则 C 点位于 ABD 的外接圆之外，可以证明右边的三角形最小内角大于左边，也意味着右边的三角剖分优于左边。这是一种普遍的情况，于是可以将 BC 边标记为非法边，在网格划分算法中遇到这种情况就需要对其进行翻转，即删除 BC、连接 AD、删除 ABC,BDC、生成 ABD,ACD 。

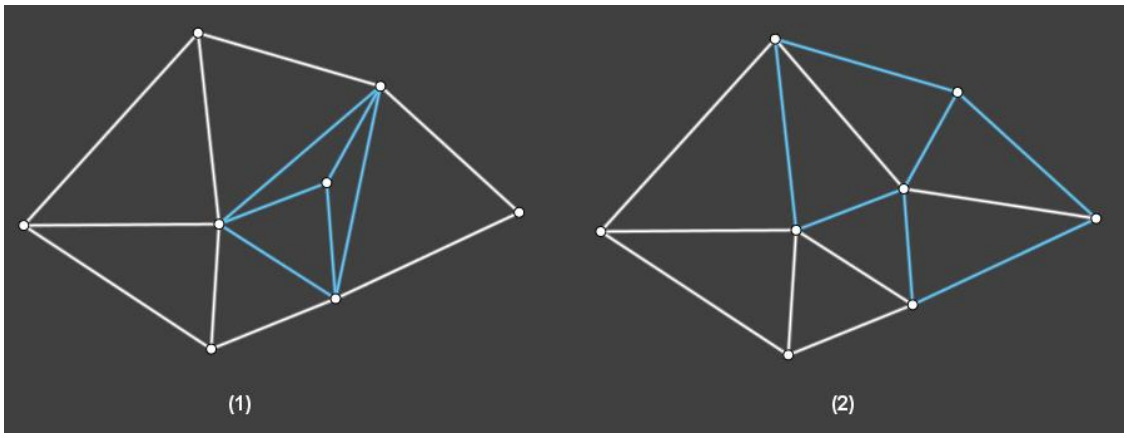
平面点集的网格划分

首先考虑一种比较简单的需求，已知平面上的一组点，现在要将其连接成网格，并满足 Delaunay 条件。随机增量算法将是一种很好的方式，思想是假设使用前 n 个点已经形成了满足条件的网格，然后插入第 n + 1 个点，增量执行这一过程，直到所有点都加入到网格中，则完成了这堆点的网格划分。现在来看看将第 n + 1 个点插入网格需要完成的操作



这里存在两种情况，位于三角形内部，或者位于三角形边上。在第一种情况下，首先找到此点所在的三角形，然后分别连接 n+1 与三角形的三个端点，这样就将新

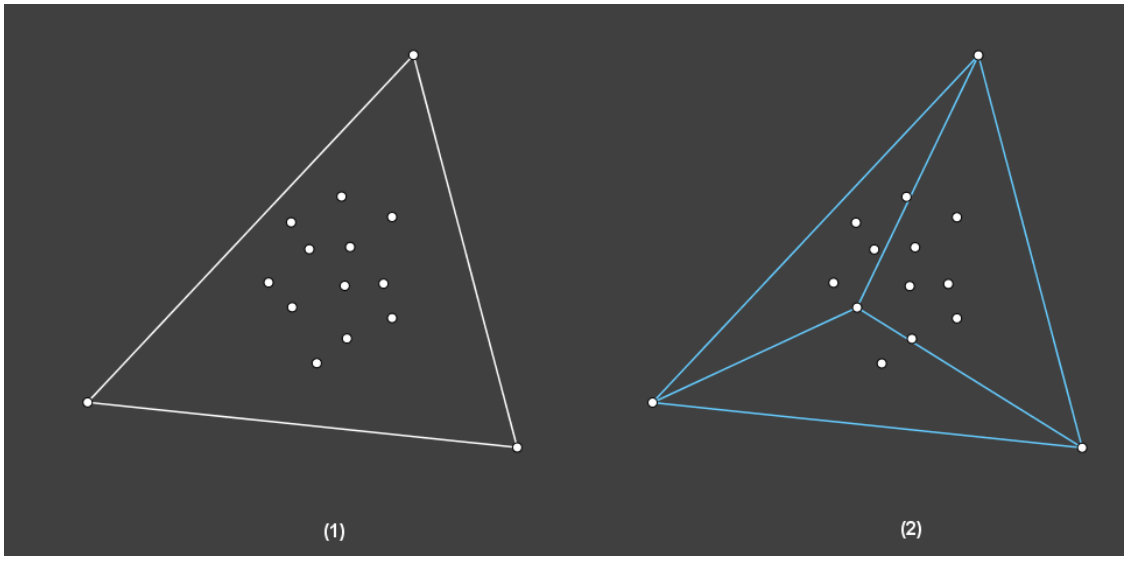
点插入到了网格中，但是很显然，除了还没检查的新生成边，原本合法的边也变成了潜在的非法边。



上图左边显示了插入一个点后，需要检查的边的情况，首先对原三角形的边进行检查并翻转非法边，得到右边的图形，由于翻转操作生成了新的三角形，又需要标记检查新的潜在非法边，持续这一过程直到所有边都合法为止。

关于初始化

前面描述了增量过程，但是在最开始我们并没有一个现成的 Delaunay 网格。一种简单的方式是用一个足够大的初始三角形将所有点包含进来，然后使用增量方式插点，完成之后再删除外围的点和线段。

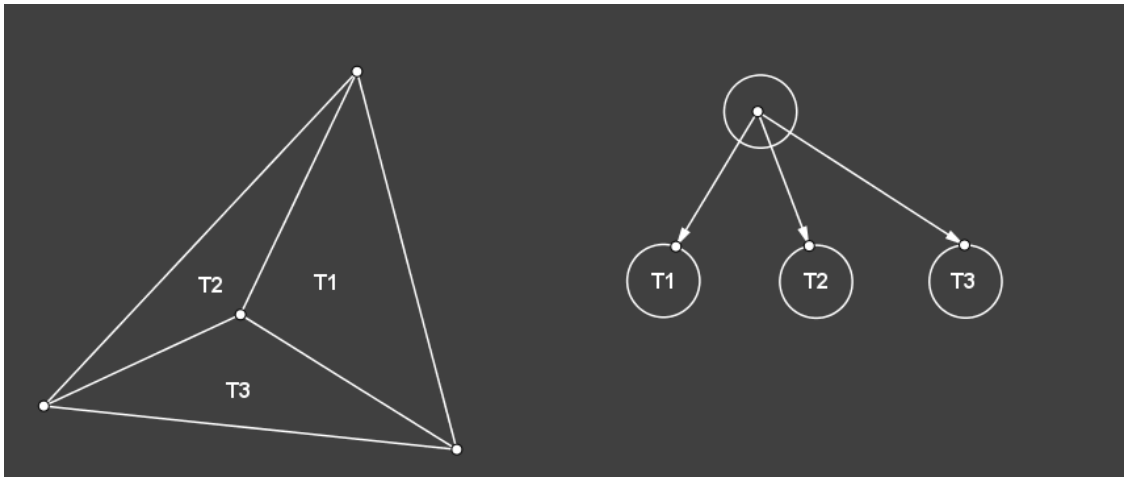


建议的初始外围三角形为 A(-4m, -4m), B(4m, 0), C(0, 4m)，其中 m 为所有点中绝对值最大的坐标值。

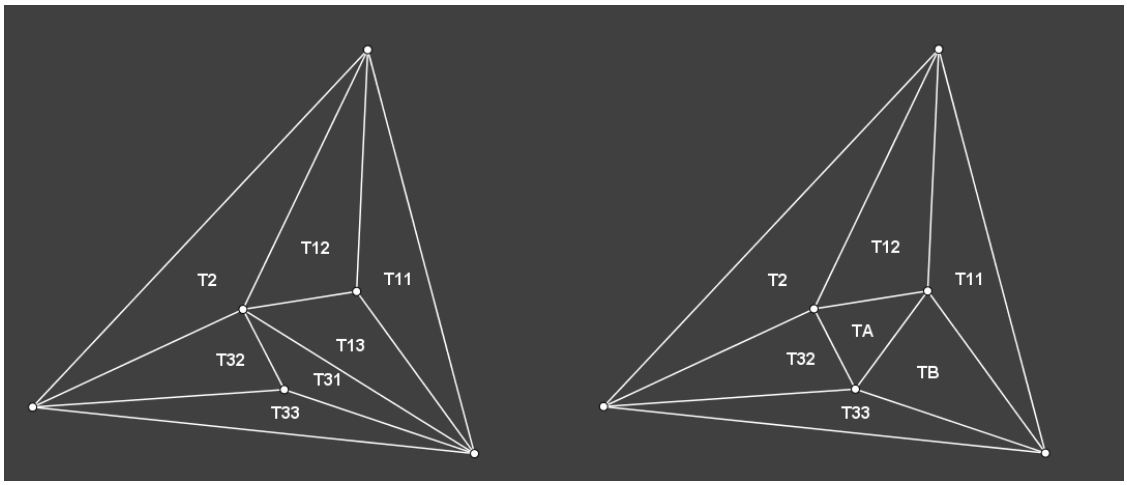
数据结构

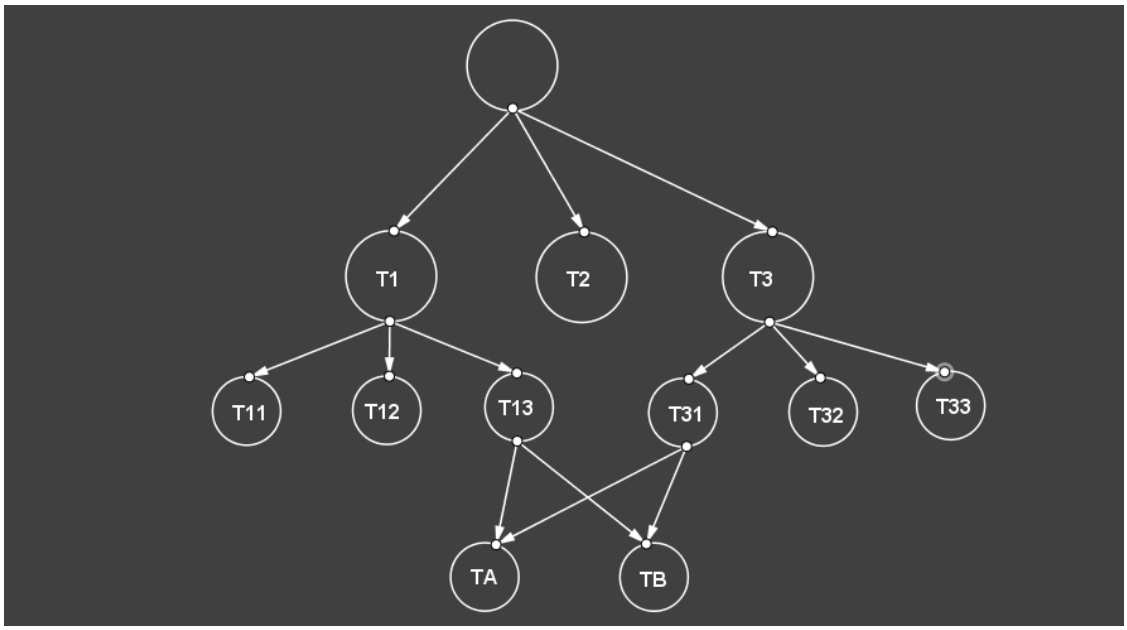
在插入点的过程中，要进行非常多的查询操作，例如查询给定点所在的三角形编号、一条边所在的一个或两个三角形编号等等，因此有必要维护一个高效的数据结构来支持这些查询。

对于三角形集合，我们使用一个有向无环图来表示所有的剖分历史，以最初的外围三角形为根节点，第一次剖分引入三个小三角形，对应于从根节点指向三个子节点



而如下图的一次翻转操作则对应用于将涉及的两个节点 T31 和 T13 指向新节点 TA 和 TB





使用这种结构，若要查询给定点所在的三角形，则从根节点开始，依次计算三个子节点，找到那个包含查询点的三角形，然后再计算它的子节点，以此类推，最终到达的叶节点即为要查询三角形。

另一方面，为了对三角形进行快速查询，我们使用数组来存储网格单元，其中每一行是三角形顶点编号，并且所有顶点也用数组来存储。

triangles:

Index	v1	v2	v3	flag
0	0	1	2	
1	1	2	3	
2	1	2	4	
…	…	…	…	

vertices:

Index	x	y
0	0	0
1	10	0
2	0	10
…	…	…

前面所说的有向无环结构存储的实际上是三角形的索引号。由于查询点的过程依赖于网格剖分历史，所以在删除单元的时候不能直接将其从数组中移除，而应该使用标记删除的方法，上面 triangles 数组中的标志 flag 位即用于此目的。

在插入点生成新的单元之后，需要检查旧单元的边是否合法，这就需要使用两个点的索引号来查询边，并找到邻边的三角形。这里仍使用数组来存储边的信息，但为了高效的通过顶点查询，我们使用搜索二叉树来存储边的索引号，搜索方式为边的中点坐标，从左到右，从下到上。

同时，边的信息应该包含其所在的一个或两个三角形，以及一个删除标志位(因为二叉树存储了边的索引号，不能直接将边从数组中移除)。

edges:

Index	v1	v2	t1	t2	flag
0	0	1	0	1	
1	1	2	1	2	
2	2	3	3	3	
...	...	...	...	...	...

非法边检测

假设相邻的两个三角形为 ABC 和 BCD，四个点的坐标分别为

$$A(a_x, a_y), \quad B(b_x, b_y), \quad C(c_x, c_y), \quad D(d_x, d_y)$$

那么可以计算得到三角形 ABC 的外接圆心坐标为

$$x = (c_1 * a_2 - c_2 * a_1)/(a_2 * b_1 - a_1 * b_2)$$
$$y = (c_1 * b_2 - c_2 * b_1)/(a_1 * b_2 - a_2 * b_1)$$

其中

$$a_1 = b_x - a_x$$
$$b_1 = b_y - a_y$$
$$c_1 = \frac{1}{2}(b_x^2 - a_x^2 + b_y^2 - a_y^2)$$
$$a_2 = c_x - b_x$$
$$b_2 = c_y - b_y$$
$$c_1 = \frac{1}{2}(c_x^2 - b_x^2 + c_y^2 - b_y^2)$$

然后再计算 D 点与圆心的距离是否大于外接圆半径就可以判断了。

OP2 中的算法

The OP2 approach to the solution of unstructured mesh problems (based on ideas developed in its predecessor OPlus [17, 19]) involves breaking down the algorithm into four distinct parts: (1) sets, (2) data on sets, (3) connectivity (or mapping) between the sets and (4) operations over sets.

OP2's general decomposition of unstructured mesh algorithms, imposes no restrictions on the actual algorithms, it just separates the components of a code.

其中还涉及数值计算的一些算法

The straightforward programming interface combined with efficient parallel execution makes it an attractive prospect for the many algorithms which fall within the scope of OP2. For example the API could be used for explicit relaxation methods such as Jacobi iteration; pseudotime-stepping methods; multi-grid methods which use explicit smoothers; Krylov subspace methods with explicit preconditioning; semi-implicit methods where the implicit solve is performed within a set member, for example performing block Jacobi where the block is across a number of PDE's at each vertex of a mesh. However, algorithms based on order dependent relaxation methods, such as Gauss-Seidel or ILU (incomplete LU decomposition), lie beyond the current capabilities of the framework.

算法的关键是迭代

The algorithm iterates towards the steady state solution, in each iteration using a control volume approach - for example the rate at which the mass changes within a control volume is equal to the net flux of mass into the control volume across the four faces around the cell.

算法举例

Algorithm

1. **for** each op_dat requiring a halo exchange {
2. **execute** CUDA kernel to gather export halo data
3. copy export halo data from GPU to host
4. **start** non-blocking MPI communication }
5. **execute** CUDA kernel with op_plan for *core* elements
6. **wait for all** MPI communications to complete
7. **for** each op_dat requiring a halo exchange
8. { copy import halo data from host to GPU }
9. **execute** CUDA kernel with op_plan for non-*core* elements

Figure 5: Multi-GPU non-blocking communication

http://book.51cto.com/art/201801/564842.htm

Software Requirements Specification

for

<Mesh-based Application>

Version 1.0 approved

Prepared by <陈灿辉 17343008>

<中山大学数据科学与计算机学院软工一班>

<2019-05-01>

Mesh-based Application

1. Introduction

1.1 Requirement Statements

We will develop a object-oriented class library (we call system) for domain users (domain programmers) to develop mesh-based applications. The class library should be a basic foundation to let domain users write and run mesh-based applications.

Mesh may be created by many different domain tools (pre-processing tools) in different formats. These formats will be as the input of our designing system. Our system usually outputs the simulation results in some user-defined formats to some post-processing tools. These tools may let us have a visualized view of our simulated results.

In a domain user's program, mesh structure is considered as a static structure. It is created firstly as a text file by certain domain tools. The system should have good compatibility with different format file, and can load/store/convert mesh structure into different format files, including mesh format in text file, hdf5 file, binary, or database file. Besides, the system should provide an application interface for others domain tools.

The system (or class library) we are designing first let user create a mesh-based application in a programming language like C/C++. In the application, user typically first loads a mesh from a text file(or others format file) into memory data structure; then defines some kernel functions over the structure; further by using these kernel functions defines an algorithm over the structure. The algorithms over mesh structure usually are loops of kernel operations over mesh. Our designed system should provide some basic functionalities to support the executions of mesh-based applications under certain platform and systems (e.g. C++ under Linux).

The system should have good compatibility with different format file, and can load/store/convert mesh structure into different format files, including mesh format in text file, hdf5 file, binary, or database file. Besides, the system should provide a template for domain user to define their own mesh directly in code. And the system should check for the mesh, and alert when something goes wrong. The system also allow domain programmer to define kernel function and algorithm in mesh-based application. In order to make the domain programmer to program conveniently, the system should provide commonly used kernel function or mesh-based algorithm. The system can speed up the computation automatically by using multi-core CPU or GPU. And the system can implement loop of kernel operations over mesh, making them parallelly running. And the system should be reliable, it can run continuously and make a backup for data when stop, and keep a log when running. The system should give a performance report to user when needed, and can analyze the performance and give a suggestion for improving the performance to the domain programmer. The detail requirements are listed as follows.

1.2 Scope of this Document

The customer and the user for the system are domain users (domain programmers) who develop the mesh-based applications, for example, CFD (Computational Fluid Dynamics). The system will be develop during the semester under the conduction of our teacher.

1.3 Overview

The system will allow domain user to load/save mesh from/to a file, declare and define the mesh data, and implement the loop of kernel operations over mesh. In our system, domain user should define the kernel operation over mesh and algorithm over mesh due to their requirement, and the system will run. And the system is also a framework for the execution of unstructured grid applications on clusters of GPUs or multi-core CPUs. And the system can improve the performance by using multi-core CPU parallelization processing, efficient numerical algorithm (typically in matlab), and GPU acceleration and so on.

1.4 Business Context

In school, we learn this project to develop our ability of system analyse and design.

[For OP2/OPS, OP2 and OPS was launched by Mike Giles at the University of Oxford. Other past and current collaborators include: Carlo Bertolli, Satya Jammy, Neil Sandham, Lawrence Mitchell, Adam Betts, David Ham, Paul Kelly,Nicolas Loriant, Graham Markall, Florian Rathgeber, Christian T. Jacobs, Jianping Meng, Attila Sulyok, Daniel Balogh, Endre László, Ben Spencer, Yoon Ho, Leigh Lapworth, David Radford, Matt Street, Massimiliano Leoni,Andrew Owenson, Stephen Jarvis, Nick Hills and others.]

2. General Description

2.1 Product Functions

for mesh-based applications.

1. Storing Mesh structure as a text file
2. Storing Mesh Structure in Memory
3. Loading mesh from a text file
4. Save Mesh to a text file from Memory
5. Defining Kernel Operations over Mesh
6. Algorithm over mesh (Mesh Application)
7. implementing loop of kernel operations over mesh
8. Domain programmer may use different mesh format in text file, hdf5 files, or binary, or database files
9. Programmer can define mesh (elements) directly in his code
10. Running on many-core architecture
11. System should be implemented in programming language C/C++
12. Performance Considerations
13. Memory should be used efficiently
14. Traces for performance
15. Input Output from or to a text file to save computation results in each stop.
16. Using database to store the mesh
17. Using GPU to speed up

2.2 Similar System Information

A mesh-based application framework is OP2. It is a programming abstraction for writing unstructured mesh algorithms, and the corresponding software library and code translation tools to enable automatic parallelisation of the high-level code. But in this system we are going to develop a similar system but simpler as the course project in SYSU system analyse and design.

2.3 User Characteristics

The domain users (domain programmers) can use our system to develop their mesh-based application easily, and speed up by using parallelisation.

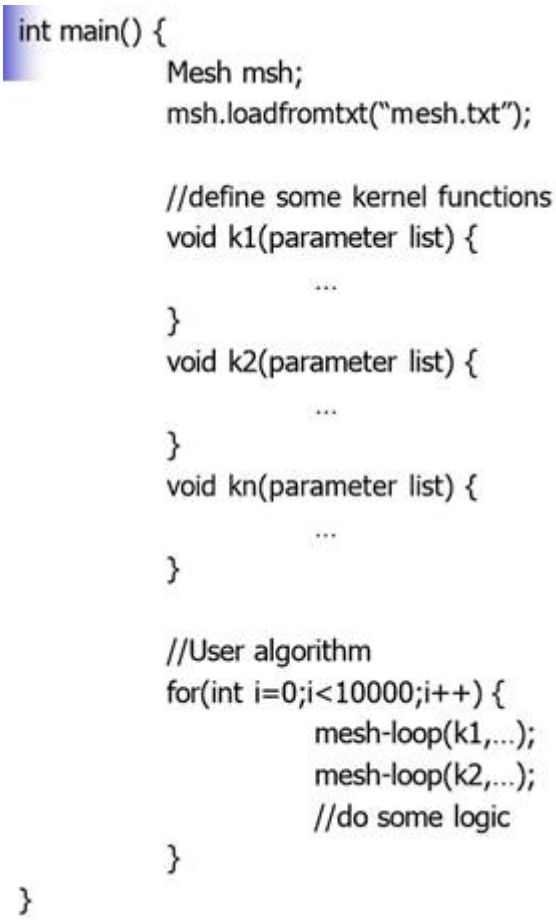
2.4 User Problem Statement

Developing a mesh-based application without any framework is difficult and the speed of calculation is quite slow without using parallelization. Besides, the persistence of the mesh and how to save and calculate economically is also a problem.

2.5 User Objectives

The user wants a database that will store information on a silent auction. The program must faciliatate the speed and ease of input. It also must store the items the IDANRV needs to store.

The users want a simple and clear framework to use without worrying about the internal details. A simple of the framework may look like.



Users can simple use this system to develop their mesh-based application with high computing speed.

New Version in C++, an user may use the library as follows

This is my preliminary design.

```
#include <meshApp.h>

```

```
/**
 * To define you own attribute
 * you should encapsulate the user-defined attribute in a class
 * and the class must inherited the abstrace class Attribute
 * and implement the virtual function
 */

class MyAttribute : public Atttribute{
    //you attribute
public:
    double temperature; //

    //...

    //implement the virtual function
    void update() {
        //you implementation

    }
}

/**
 * To deinfe you own operation on mesh
 *  you should encapsulate the user-defined opeartion in a class
 * and the class must inherited the abstrace class MeshOperation
 * and implement the virtual function
 */

class MyOpeartion : public MeshOperation {
public:

    double updateCell(Cell * cell) {

    }

    double updateDomain(Domain * domain) {

    }

    double updateEdge(Edge * edge){

    }

    double updateFace(Face * face) {

    }

    double updateMesh(mesh * mesh){

    }

    double updateNode(Node * node){

```

```
    }

    double updateZone(Zone * Zone) {

    }
}

/**
 * To start the loop
 * you should encapsulate the loop algorithm in a class
 * and the class must inherited the abstrace class Loop
 * and implement the virtual function
 */

class MyLoop : public Loop{
public:
    virtual void loopAlgorithm(){
        //you implementation here
        //...
    }

    void setOperation(MeshOpeartion * operation){
        //..
    }

    void setParallelSystem(){
        //...
    }

}

int main(){
    //first load mesh
    meshApp* myMeshApp = new meshApp();
    myMeshApp->loadmesh("..."); //load mesh from file

    Loop * loop = new MyLoop();
    loop->run();

    //get report
    myMeshApp->report();

    myMeshApp->savemesh("..."); //store mesh into file

    delete(myMeshApp);
    delete(loop);
}
```

2.6 General Constraints

The system will be developed by using C/C++, and inorder to speed up the algoithm, the user should have muli-core CPU or NVIDIA CUDA.

3. Functional Requirements

3.1 Storing Mesh structure as a text file

1. The system should store Mesh structure as a single text file
2. The system can split the text file if the Mesh structure is too complicated, the text file is too large
3. The system should have the property that the text file should be readable can be compatible with multiple file formats

3.2 Storing Mesh Sturcture in Memory

1. The system should store mesh structures in memory using set and map structures.
2. The system should have data structures that should support efficient operations. All of the numerically-intensive operations can then be described as a loop over all members of a set, carrying out some operations on data associated directly with the set or with another set through a mapping
3. The system's data structure should not cost too many space
4. take op2 as an example

```
typedef struct {
  int index;        /* index */
  int size;         /* number of elements in set */
  char const *name; /* name of set */
                    // for MPI support
  int core_size;    /* number of core elements in an mpi process*/
  int exec_size;    /* number of additional imported elements to be executed */
  int nonexec_size; /* number of additional imported elements that are not
                       executed */
} op_set_core;
```

```
typedef struct {
  int index;        /* index */
  op_set from,      /* set pointed from */
     to;            /* set pointed to */
  int dim,          /* dimension of pointer */
     *map;          /* array defining pointer */
  int *map_d;       /* array on device */
  char const *name; /* name of pointer */
  int user_managed; /* indicates whether the user is managing memory */
} op_map_core;
```

   the structure should be clear and useful

3.3 Loading Mesh from a text file

1. The loading should compatible with multiple file formats
2. The system will alert you if something goes wrong
3. The system mush check the mesh structure when loading
4. The system can load Partial loading when the file is too large and out of memory

3.4 Save Mesh to a text file from Memory

1. The system can store mesh structure as a single text file
2. The system can split the text file if the mesh structure is too complicated, the text file is too large.
3. The system generated text file should be readable can be compatible with multiple file formats

3.5 Defining Kernel Operations over Mesh

1. The system should allows users to customize functions as kernel operation
2. The system should Fix the kernel function format, e.g. fixed parameters
3. The systems should provide some commonly used kernel operation for users, adjusting the operation by passing different parameters
4. The system should have the constraint that the operations can be a function or a structure/class
5. The system will throw an exception when something wrong
6. A simple kernel Operation may look like

```
inline void update(const double *qold, double *q, double *res,
                   const double *adt, double *rms) {
  double del, adti;

  adti = 1.0f / (*adt);

  for (int n = 0; n < 4; n++) {
    del = adti * res[n];
    q[n] = qold[n] - del;
    res[n] = 0.0f;
    *rms += del * del;
  }
}
```

3.6 Algorithm over mesh (Mesh Application)

1. The system should allows users to customize functions as Algorithm
2. The system should fix the algorithm function format, e.g. fixed parameters
3. The system should provide some commonly used algorithm for users, adjusting the operation by passing different parameters
4. The system should have the constraint that the operations can be a function or a structure/class
5. The system should throw an exception when something wrong
6. The system should provide an interface to use the numerical algorithm in Matlab or Fortran

3.7 Running on many-core architecture

1. The system should check hardware status and configuration
2. The system should allow users to customize the number of core to be use
3. The system should have the default setting is to use all the cores to compute

3.8 Performance Considerations

1. The system can evaluate the cost time before running the application
2. The system can adjust the evaluating time by different parameters given by user
3. The system can give a suggestion to improve the performance after evaluating

3.9 Traces for performance

1. The system can evaluate the performance in terms of running time and the memory usage and so on
2. The system can give a report to users when finishing
3. The system can give a suggestion to improve the performance after evaluating

3.10 Input Output from or to a text file to save computation results in each stop

1. The system should save the computation results in each stop
2. The system should maintenance a log while running
3. The system should recover data and continue the computation after restarting

3.11 Using database to store the mesh

1. The system should store and load the mesh from/into the DB
2. The system should support the mainstream data base

3.12 Using GPU to speed up

1. The system should check hardware status and configuration
2. The system should support the mainstream GPU like NVIDIA
3. The system's the default setting is to use GPU

3.13 Defining mesh directly in code

1. The system should provide a framework(template) for use to define their own mesh in code directly
2. The system should check for the correctness for the user-defined mesh

4. Interface Requirements

4.1 User Interfaces

The system should have as least the following interface

1. GUI. A simple GUI for use to use with users' guide
2. CLI. It also allow users to use command line to run the system
3. API. It should provide the API for another software

4.2 Hardware Interfaces

1. The system should provide a multi-core CPU interface
2. The system should provide a GPU interface

4.3 Communication Interfaces

1. The system should support distributed computing via network

4.4 Software Interfaces

1. The system can import and export data with database and another app like CAD

5. Performance Requirements

(Take the following as an example)

500 MHz processor or higher

256MB RAM or higher

1.5GB Available Hard Drive Space

6. Non-function Requirements

6.1 Security

1. If the system use data base for storage, there is a requirement that could be implemented are encrypting the database and/or making the database password-protected, by user’s request.
2. The system should include all available safeguard from viruses deleting the data

6.2 Performance

1. The system should give a result of meshing within one minute when the mesh is not so complicate
2. The system can work continuously for a long time (at least 500 hours) without any stop

3. The system should give a feedback to the user'request (such as stopping, asking for report) within one second even in calculating
4. The system should have high throughput, allowing multiple parallel operations to be performed simultaneously(e.g. more than 100 parallel process at the same time)
5. The system should allow the operation of high-precision floating-point Numbers
6. The system should make back-ups regularly so that restoration with minimal data loss is possible in the event of unforeseen events.
7. The system should use the memory efficiently

6.3 Cultural and Political

1. The system should provide user manual in different language
2. The system should following the legal requirement in the release country
3. The system should follow international standards

6.4 Operational

1. The system should support the programming language C/C++
2. The system should work well in Linux OS
3. The system should make the good use of multi-core CPU
4. The system can use the GPU to speed up
5. This system should be compatible with any computer using mainstream CPU and GPU

6.5 Compatibility

1. The system should be compatible with different version of the software, OS, database
2. The system should allow user to choose different version of the system
3. The released system should have a long-term maintenance

6.6 Usability

1. The system can modify the configuration through the configuration file
2. The system should provide users with different examples to use the system
3. The system should provide an internationalization documentation

7. Operational Scenarios

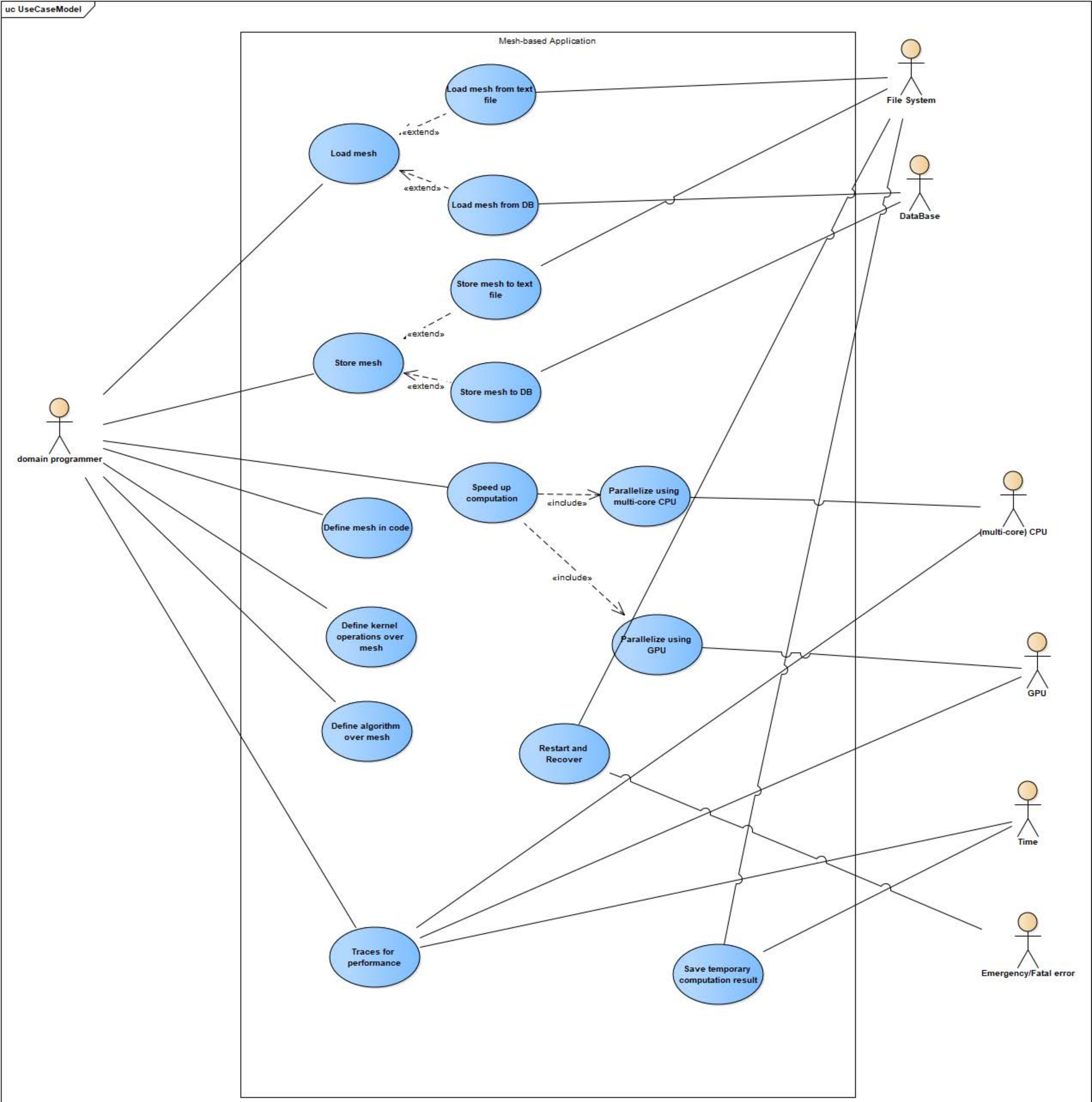
Scenario A: CFD

Computational fluid dynamics (**CFD**) is a branch of fluid mechanics that uses numerical analysis and data structures to analyze and solve problems that involve fluid flows. And our system can improves the accuracy and speed of complex simulation scenarios.

Scenario B: CAD

Computer-aided design (**CAD**) is the use of computers (or workstations) to aid in the creation, modification, analysis, or optimization of a design. Our system should be able to load mesh from CAD or other application.

uc UseCaseModel



Use case: Load mesh
ID: UC1
Actor: domain programmer, File System, Database
Descriptions: The domain programmer can load the mesh from file or database
Preconditions: The system is ready and can access to file or database
Flow of events: 1. The programmer defines the structure(schema) of the mesh before loading 2. The programmer calls the API of loading mesh 3. The system finds the source file for loading 4. If the system can access to the mesh file 4.1. The system checks the structure of mesh 4.2. If the structure is valid 4.2.1. The system loads the mesh to memory 4.3. Else 4.3.1. The system throws an exception 5. Else 5.1. The system throws an exception
Postconditions: The mesh has loaded in the memory
Alternative flow: 1. The file is invalid or inaccessible 2. The mesh structure does not match the content

Use case: Load mesh from text file
ID: UC2
Parent use case ID: UC1
Actor: domain programmer, File System
Descriptions: The domain programmer can load the mesh from text file
Preconditions: The system is ready and can access to file
Flow of events: *1. The programmer defines the structure(schema) of the mesh before loading* *2. The programmer calls the API of loading mesh* 3. The system finds the source file for loading 4. If the system can access the mesh file 4.1. The system checks the structure of mesh 4.2. If the structure is valid 4.2.1 The system loads the mesh to memory 4.3. Else

4.3.1. The system throws an exception 5. Else 5.1. The system throws an exception
Postconditions: The mesh has loaded in the memory
Alternative flow: 1. The file is invalid or inaccessible 2. The mesh structure does not match the content

Use case: Load mesh from DB
ID: UC3
Parent use case ID: UC1
Actor: domain programmer, Database
Descriptions: The domain programmer can load the mesh from Database
Preconditions: 1. The system is ready 2. The database server is accessible 3. The domain programmer have access to the database
Flow of events: *1. The programmer defines the structure(schema) of the mesh before loading* **2. The programmer accesses to the database** *3. The programmer calls the API of loading mesh* 4. If the system can access the mesh file 4.1. The system checks the structure of mesh 4.2. If the structure is valid 4.2.1 The system loads the mesh to memory 4.3. Else 4.3.1 The system throws an exception 5. Else 5.1. The system throws an exception
Postconditions: The mesh has loaded in the memory
Alternative flow: 1. The database is inaccessible 2. The mesh structure does not match the content

Use case: Store mesh
ID: UC4
Actor: domain programmer, File system, Database
Descriptions: The domain programmer can store mesh structure from memory into file or database
Preconditions: 1. The mesh structure is correct and complete. 2. The operation on the mesh structure is stopped or finished at that time
Flow of events: 1. The programmer chooses a mesh structure in memory to store. 2. The programmer chooses the format of mesh to store 3. The programmer calls the API to store 4. The system checks the accessibility of the file/database 5. If the file/database is accessible 5.1. The system stops the operation on the mesh and store it in a special format 6. ELSE 6.1. The system throws an error
Postconditions: The mesh has stored
Alternative flow: 1. The file/database is inaccessibly 2. The file is too big, need to be spited into some smaller

Use case: Store mesh into text file
ID: UC5
Parent use case ID: UC4
Actor: domain programmer, File system
Descriptions: The domain programmer can store the mesh into text file
Preconditions: 1. The mesh structure is correct and complete. 2. The operation on the mesh structure is stopped or finished at that time
Flow of events: *1. The programmer chooses a mesh structure in memory to store.* *2. The programmer chooses the format of mesh to store* *3. The programmer calls the API to store* 4. The system checks the accessibility of the file 5. If the file is accessible 5.1. The system stops the operation on the mesh and store it in a special format 6. ELSE 6.1. The system throws an error
Postconditions:

The mesh has stored into file(s)
Alternative flow: 1. The path of the file is incorrect 2. The mesh structure does not match the content

Use case: Store mesh into database
ID: UC6
Parent use case ID: UC4
Actor: domain programmer, Database
Descriptions: The domain programmer can store the mesh into text file
Preconditions: 1. The mesh structure is correct and complete. 2. The operation on the mesh structure is stopped or finished at that time
Flow of events: *1. The programmer chooses a mesh structure in memory to store.* *2. The programmer chooses the format of mesh to store* *3. The programmer calls the API to store* 4. The system checks the accessibility of the database 5. If the database is accessible 5.1. The system stops the operation on the mesh and store it into the database 6. ELSE 6.1. The system throws an error
Postconditions: The mesh has stored into the database
Alternative flow: 1. The database is inaccessibly 2. Exceptions may happen during the connecting / accessing to the database

Use case: Speed up computation
ID: UC7
Actor: domain programmer, (multi-core) CPU, GPU
Descriptions: The system can check hardware statues and speed up the computation and parallelize the loop of operations automatically or manually.
Preconditions: 1. The hardware meets the performance requirements 2. The domain programmer has enabled this function
Flow of events: 1. The programmer customize the configuration 2. The system should check hardware status and configuration 3. If the hardware meets the least requirement

3.1. Run the program parallelly 4. ELSE 4.1. The system gives out a warning, and does not use the hardware to speed up
Postconditions: The program can run parallelly.
Subcases: Parallelize using multi-core CPU, Parallelize using GPU
Alternative flow: 1. The hardware statues does not meet the requirement

Use case: Parallelize using multi-core CPU
ID: UC8
Actor: domain programmer, (multi-core) CPU
Descriptions: The system can check hardware statues and speed up the computation and parallelize the loop of operations automatically or manually.
Preconditions: 1. The hardware meets the performance requirements 2. The domain programmer has enabled this function
Flow of events: 1. The programmer customize the number of core to be use 2. The system should check hardware status and configuration 3. If the hardware meets the least requirement 3.1. Run the program parallelly 4. ELSE 4.1. The system gives out a warning, and does not use the hardware to speed up
Postconditions: The program can run parallelly.
Alternative flow: 1. The hardware statues does not meet the requirement

Use case: Parallelize using GPU
ID: UC9
Actor: domain programmer, GPU
Descriptions: The system can check hardware statues and speed up the computation and parallelize the loop of operations automatically or manually.
Preconditions: 1. The hardware meets the performance requirements 2. The domain programmer has enabled this function
Flow of events: 1. The programmer decide whether to use GPU or not 2. The system should check hardware status and configuration

3. If the hardware meets the least requirement 3.1. Run the program parallelly 4. ELSE 4.1. The system gives out a warning, and does not use the hardware to speed up
Postconditions: The program can run parallelly.
Alternative flow: The hardware statues does not meet the requirement

Use case: Define mesh in code
ID: UC10
Actor: domain programmer
Descriptions: The programmer can define mesh structure in code directly
Preconditions: N/A
Flow of events: 1. The programmer use a framework(template) provided by the system to define their own mesh in code directly 2. The system will check for the correctness for the user-defined mesh 3. If the mesh is not valid, the system will throw an exception
Postconditions: The mesh structure is defined by programmer
Alternative flow: The mesh structure is not valid

Use case: Define kernel operations over mesh
ID: UC10
Actor: domain programmer
Descriptions: The programmer can define kernel operation in code directly
Preconditions: N/A
Flow of events (Primary scenarios): 1. The programmer define a kernel operation in a function or structure format 2. The system will check if the operation meets the constraint, like parameters list, return value, and so on. 3. If the operation does not meet the constraint, the system will throw an exception
Postconditions: The kernel operation is defined by programmer
Secondary scenarios: Use the kernel operation offered by the system
Alternative flow: The kernel operation is not valid

Use case: Define kernel operations over mesh **Secondary scenarios:** Use the kernel operation offered by the system
ID: UC10
Actor: domain programmer
Descriptions: The programmer can use the kernel operation offered by the system
Preconditions: The system should install the corresponding libraries
Flow of events (Secondary scenarios): 1. The programmer include the corresponding libraries of the kernel operations provided by the system or the third party 2. The system will check the validation of the corresponding libraries 3. If the libraries are not accessibly, the system will throw an exception 4. The programmer using the system kernel operations by passing parameters 5. The system will check for the validation of the parameters list 6. If the parameters list does not meet the constraint, the system will throw an exception
Postconditions: The kernel operation is defined/used by programmer
Alternative flow: The kernel operation is not valid The libraries are not valid The parameters list is not valid

Use case: Define algorithm over mesh
ID: UC11
Actor: domain programmer
Descriptions: The programmer can define algorithm over mesh
Preconditions: N/A
Flow of events (Primary scenarios): 1. The programmer define an algorithm in a function or structure format 2. The system will check if the algorithm meets the constraint, like parameters list, return value, and so on. 3. If the operation does not meet the constraint, the system will throw an exception
Postconditions: The algorithm is defined by programmer
Secondary scenarios: Use the algorithm offered by the system
Alternative flow: The algorithm is not valid

Use case: Define algorithm over mesh **Secondary scenarios:** Use the algorithm offered by the system
ID: UC11
Actor: domain programmer
Descriptions: The programmer can use the kernel operation offered by the system
Preconditions: The system should install the corresponding libraries
Flow of events (Secondary scenarios): 1. The programmer include the corresponding libraries of the algorithm provided by the system or the third party 2. The system will check the validation of the corresponding libraries 3. If the libraries are not accessibly, the system will throw an exception 4. The programmer using the system algorithm by passing parameters 5. The system will check for the validation of the parameters list 6. If the parameters list does not meet the constraint, the system will throw an exception
Postconditions: The algorithm is defined/used by programmer
Alternative flow: The algorithm is not valid The libraries are not valid The parameters list is not valid

Use case: Trace for performance
ID: UC12
Actor: domain programmer
Descriptions: The programmer can trace for performance of the system
Preconditions: The system has finished at least a stage of calculation
Flow of events: 1. The user require a performance report from the system 2. The system evaluates the performance in terms of running time and the memory usage and so on 3. The system gives a report to users when finishing 4. The system gives a suggestion to improve the performance after evaluating
Postconditions: The performance report is offered to the programmer by the system
Alternative flow: The computation has not stopped

Use case: Save temporary computation result
ID: UC13
Actor: Time, File system
Descriptions: The system will save the temporary computation result at a certain period of time
Preconditions: The system has continually computed for a certain period of time
Flow of events: 1. Every once in a while, a temporary file is used to record the current calculation results 2. The system maintenances a log while running 3. If too many temporary files are generated, the system will clear the previous calculation results to save space
Postconditions: The temporary computation result is saved in file
Alternative flow: Lack of space to save the temporary computation result

Use case: Restart and Recover
ID: UC14
Actor: emergency/fatal error, File system
Descriptions: The system will stop when emergency or fatal error occurs, and restarts and recover the temporary computation result from the file
Preconditions (trigger): emergency or fatal error occurs The temporary computation result was saved in file
Flow of events: 1. When the emergency or fatal error occurs, the system is shut down unexpectedly 2. The system will restart 3. The system will give a report and try to find out the problem which cause its shut down according the log file 4. The system will read the temporary file to recover the last time temporary computation result 5. The system will ask to programmer to report the error to us (developer) to help us fix the bug
Postconditions: The system restart and recover the data
Alternative flow: The system could not find the temporary file

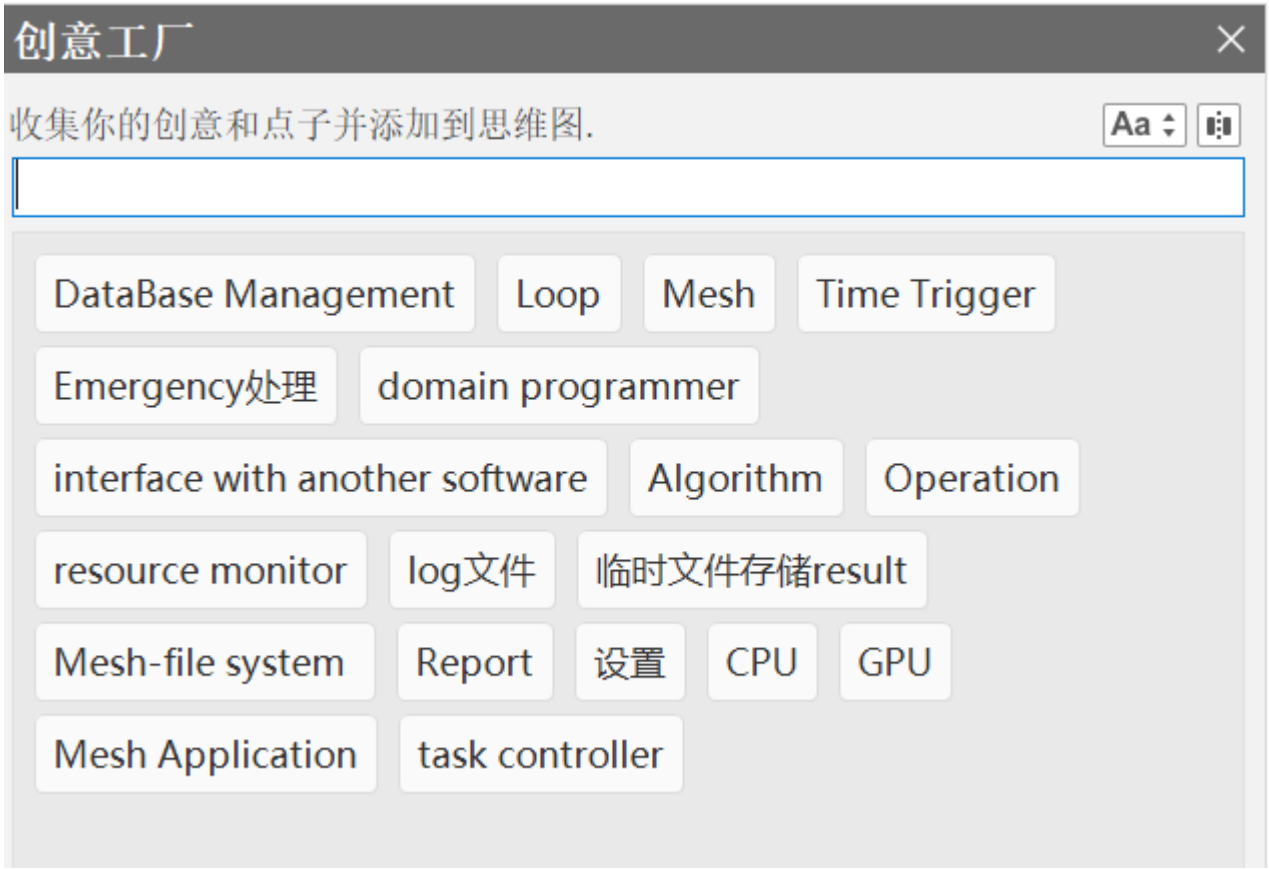
[Mesh-based Applications] Structural Modeling

Name: 陈灿辉 **studentID** : 17343008 **class**: 软工一班

Identify Analysis Classes, using Noun/Verb Analysis and CRC cards, develop a class model for mesh-based application domain.

Noun/Verb Analysis and Brainstorming

I read the use case and requirement I wrote before and use the noun/verb analysis and brainstorming to find out the class.[Of course, the brainstorm is done by myself alone]. Here I list the idea as follow.



Analyze Information

I divide the ideas listed above into 2 classes

core class:

Database Management, loop, mesh, time trigger, emergency handler, Algorithm, Operation, resource monitor, log file, temporary result file, mesh-file system, report, configuration file, Mesh application, task controller

Undecided class:

domain programmer, CPU, GPU, interface with another software

CRC Cards

I write CRC Cards in the following format.(Little different from the format shown in the teacher's courseware, lacking of superclass and subclass, because most of the classes followed have no inheritance relationship)

Database management

Responsibilities	Collaborators	Object Think	Property
Connect to the local/remote database	database, network	I know the how to connect to database	database thing
Load mesh from database into memory	Mesh, resource monitor,database	I know how to load mesh into memory	Mesh structure
Store mesh into database	Mesh,database	I know how to store mesh from memory into database	Mesh structure
Close the connection to database	database, network	I know how to close the connection	database thing

Loop

Responsibilities	Collaborators	Object Think	Property
Select mesh structure, algorithm, operation over mesh	Mesh, operation, algorithm	I know the parameter	parameter
Define an individual loop/ can run parallelly	Mesh, task controller	I am individual, and will be handle parallelly	parallelism

Mesh

Responsibilities	Collaborators	Object Think	Property
Create mesh element	mesh element, face, edge...	I know the basic element of the mesh structure	basic mesh element
Connect the element by efficient data structure	mesh element	I manage the mesh element in a special data structure	data structure
Travel the mesh element	mesh element, data structure	I know how to travel the mesh	operation
Access and update the mesh information	mesh element, data structure	I can access to the specified mesh element	operation

Time trigger

Responsibilities	Collaborators	Object Think	Property
Get the system time	system	I know the system time	time
Save the temporary computation result	mesh, result file	I know the computation result	data result
Monitor the time spent by CPU, GPU	CPU, GPU	I know the time spent by CPU and GPU	time
Keep a log	log file	I know when to keep the log file	file

Emergency Handler

Responsibilities	Collaborators	Object Think	Property
Keep a record when emergency occur	log file	I know the record	record
Recover the computation data	result file	I know the result file	file
Request a report to developer	report, network	I know the report	report

Algorithm

Responsibilities	Collaborators	Object Think	Property
Provide the commonly-use algorithm	library	I know the algorithm	algorithm
Allows users to customize functions/structure as Algorithm	mesh	I know the customize constraint	algorithm
Provide an interface to use the numerical algorithm in Matlab or other language	algorithm interface	I know interface	interface

Operation

Responsibilities	Collaborators	Object Think	Property
Provide the commonly-use operation	library	I know the operation	operation
Allows users to customize functions/structure as Algorithm	mesh	I know the customize constraint	operation
Update the mesh information	mesh	I know mesh information	mesh structure
Give the temporary/final result	CPU, GPU, result file	I know the result	data

Resource monitor

Responsibilities	Collaborators	Object Think	Property
Allocate memory/CPU/GPU resource	memory,CPU,CPU	I know the resource in computer	resource
Keep a tracer for each resource	resource, performance report	I know the situation of resource	tracer
Recycle if the resource if its not needed	OS	I know the resource	resource

Mesh application

Responsibilities	Collaborators	Object Think	Property
Initialize the subsystem	configuration file, subsystem include database management, file system, mesh, time, emergency handler, algorithm, operation	I know the subsystem	subsystem
Provide the interface for user to customize	subsystem include database management, file system, mesh, time, emergency handler, algorithm, operation	I know the interface	interface
Run the program parallelly and automatically	subsystem include database management, file system, mesh, time, emergency handler, algorithm, operation	I know the subsystem	subsystem

Task controller

Responsibilities	Collaborators	Object Think	Property
Manager the order of thread/program	Loop, CPU, GPU	I know the how to manage the order	order
Decide the maximum process/thread running at the same time	Loop, configure file	I know the loop and parallelism	parallelism
Give a report when finish a task	report	I know the report	report

mesh-file system

Responsibilities	Collaborators	Object Think	Property
Create file	disk, OS	I know the how to create file	file
Load mesh from file into memory	Mesh, resource monitor	I know how to load mesh into memory	Mesh structure
Store mesh into file	Mesh,file	I know how to store mesh from memory into file	Mesh structure
Convert the format of mesh file	mesh	I know different kind of mesh file format	format
Spite the file when it is too large	None	I know the size of the file	file size

log file

Responsibilities	Collaborators	Object Think	Property
Create file	disk, OS	I know the how to create file	file
Keep a log for each operation	operation, time trigger	I know the operation I know the time	String time

Temporary result file

Responsibilities	Collaborators	Object Think	Property
Create file	disk, OS	I know the how to create file	file
Write the temporary result into a file	Task controller, time trigger	I know the result I know the time	data(double) time
Delete the old temporary result file if not needed	time	I know the time	time

Report

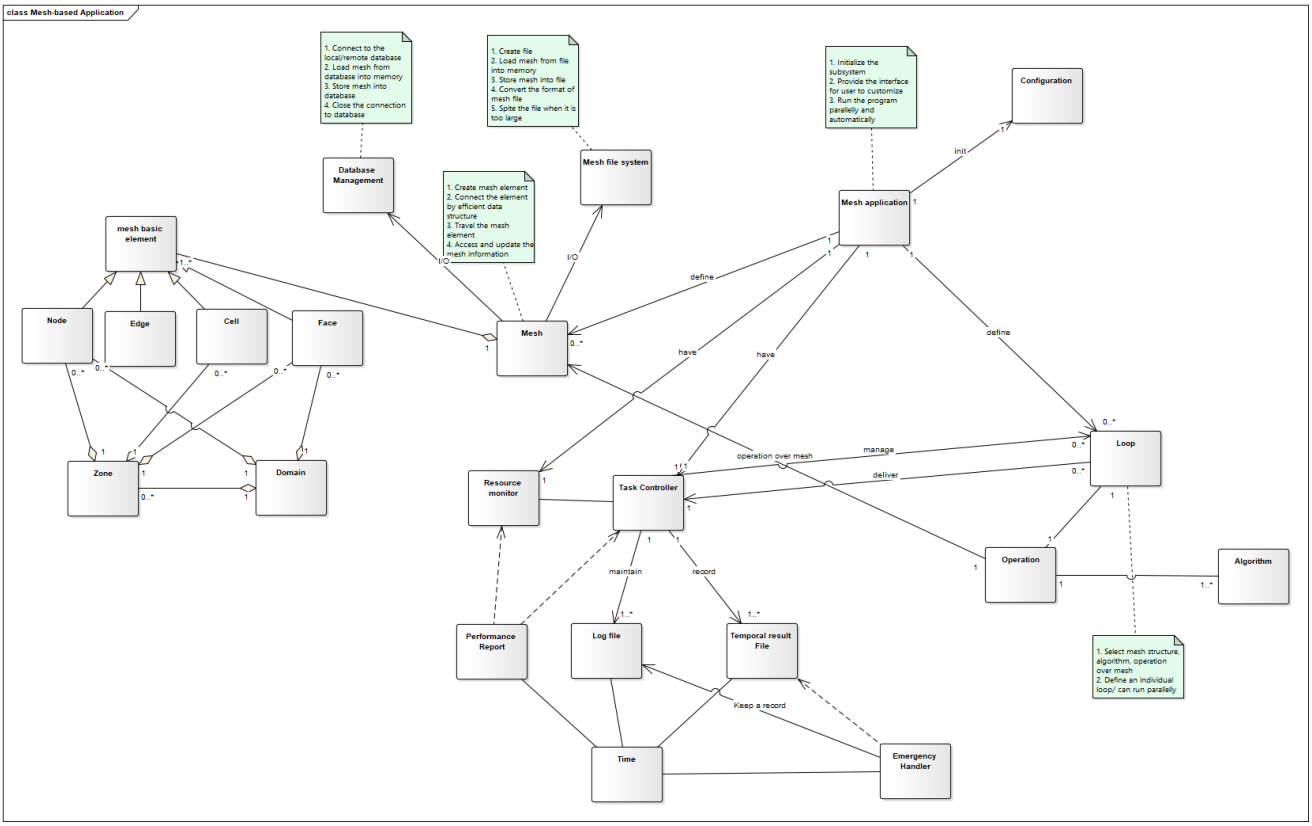
Responsibilities	Collaborators	Object Think	Property
Create file	disk, OS	I know the how to create file	file
Give a performance report	time trigger, task controller, resource monitor	I know the time spent I know the memory utilization I know the CPU/GPU utilization	time data(double) data(double)

Configuration file

Responsibilities	Collaborators	Object Think	Property
Set up default configuration	None	I know the the default configuration	configuration
Allow user to customize the configuration	domain user	I know the user	user
Check the correctness	None	I know the template of the configuration	configuration

Class Model

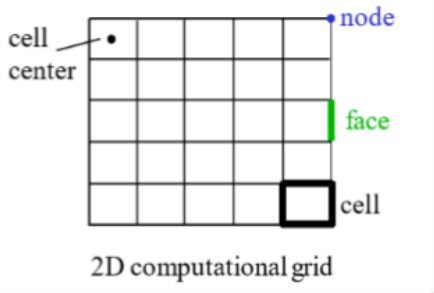
the class model for the whole system are showed as follow, and I divide the whole system in serval subsystem



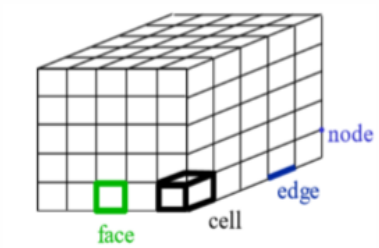
class model for mesh

Mesh Cells

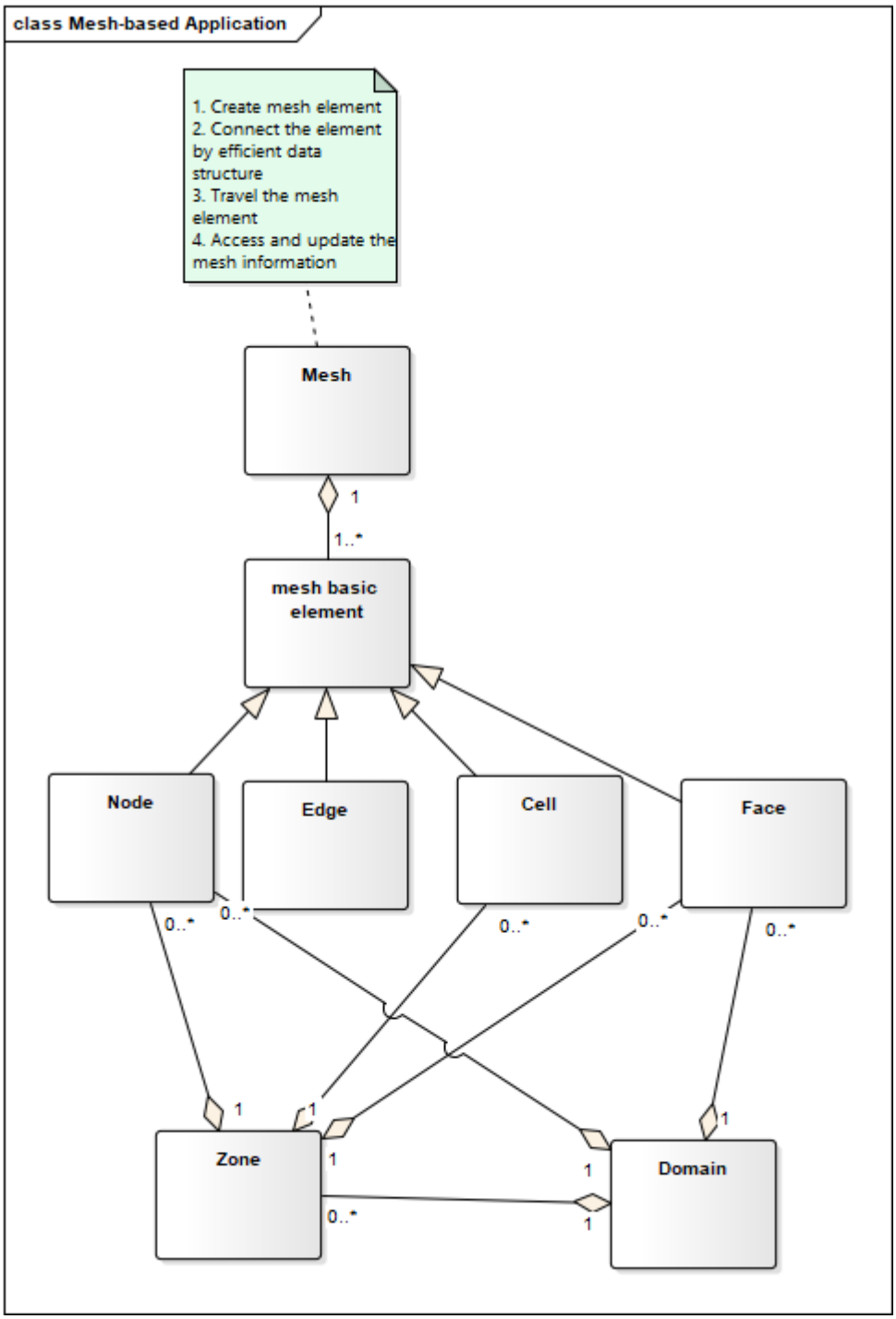
- **Node:** grid point
- **Edge:** boundary of a face
- **Cell:** control volume into which domain is broken up
- **Face:** boundary of a cell
- **Zone:** grouping of nodes, faces, and cells
- **Domain:** group of node, face and cell zones



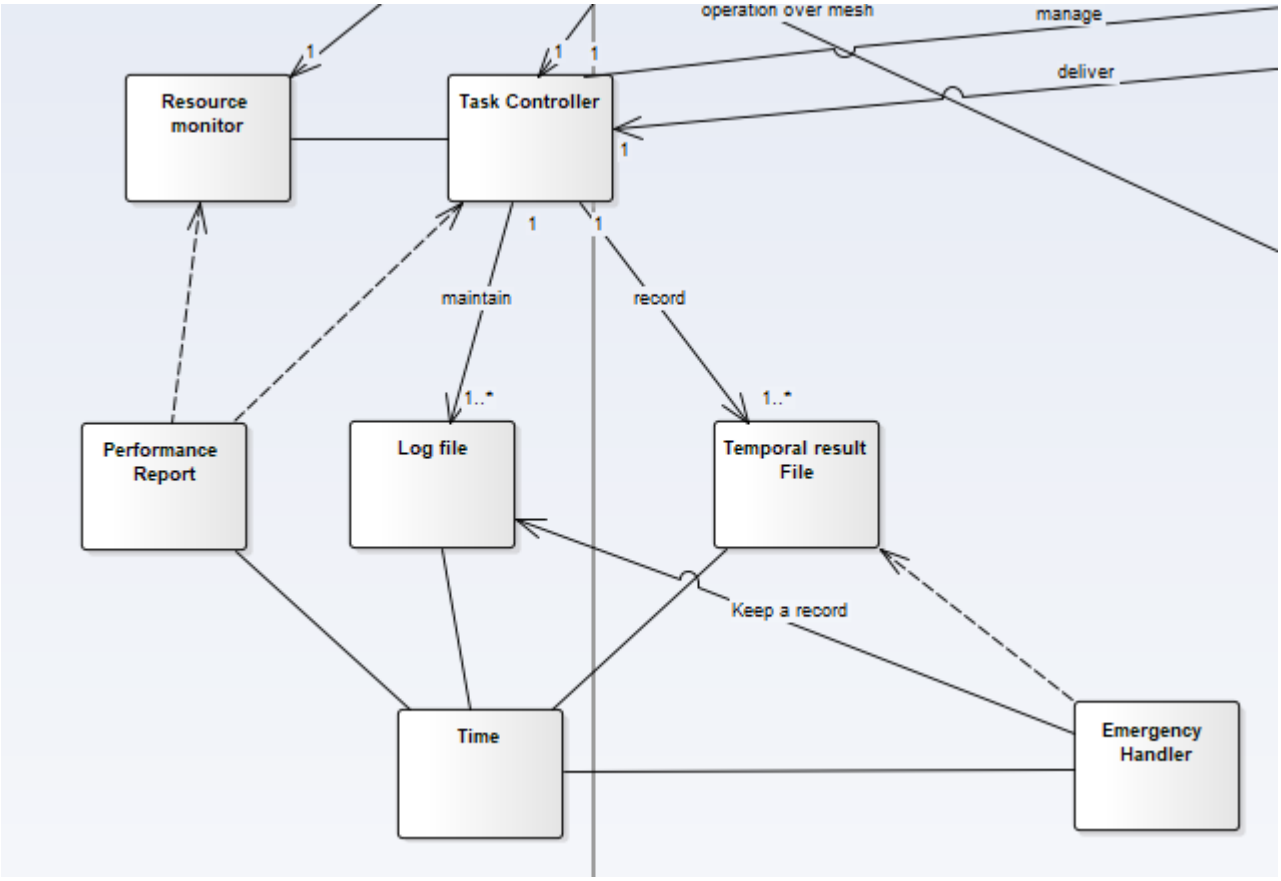
2D computational grid



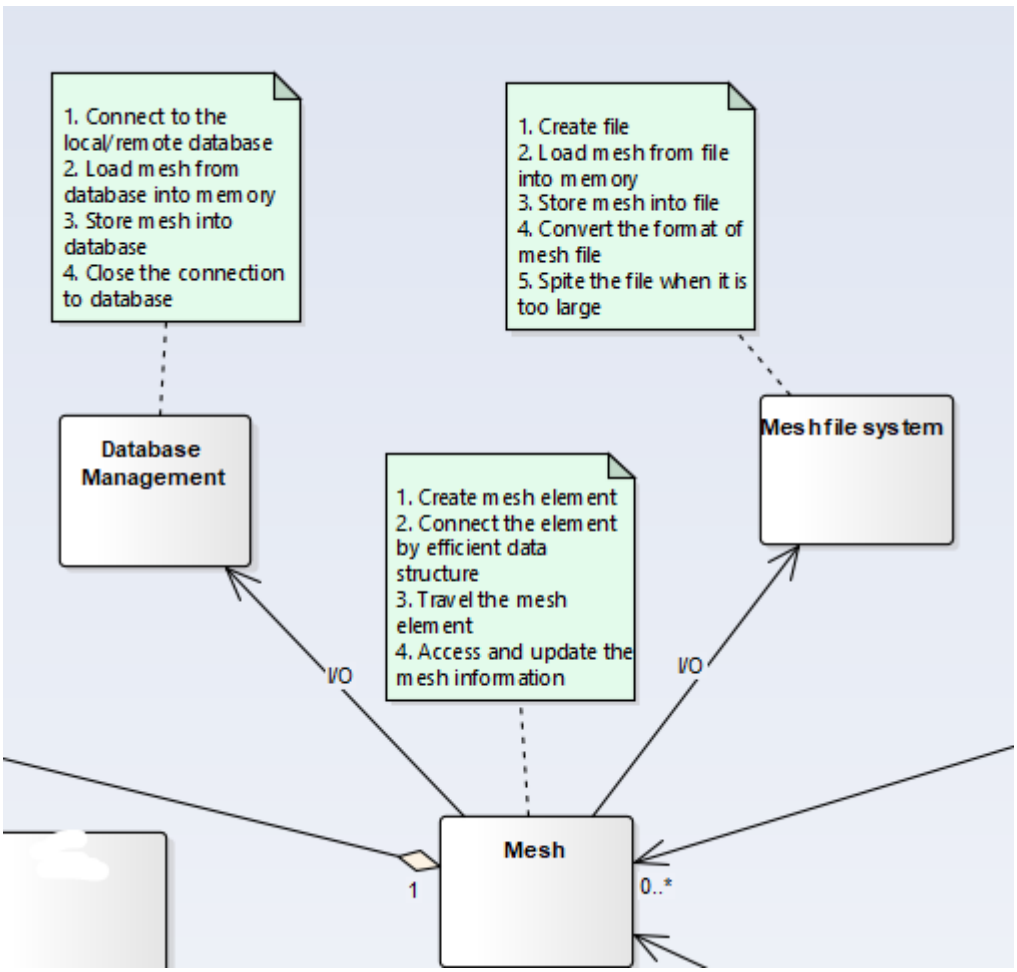
3D computational grid



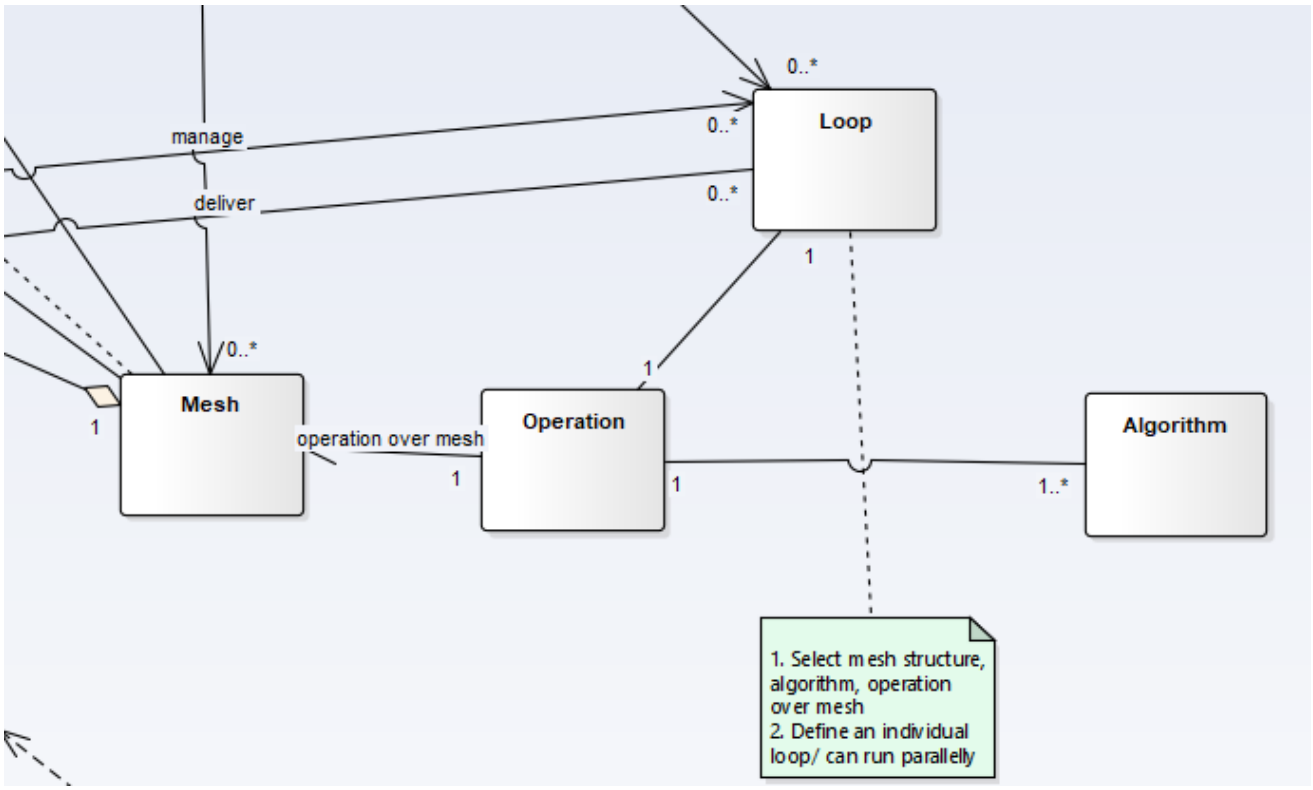
class model for record



class model for I/O



class model for running loop



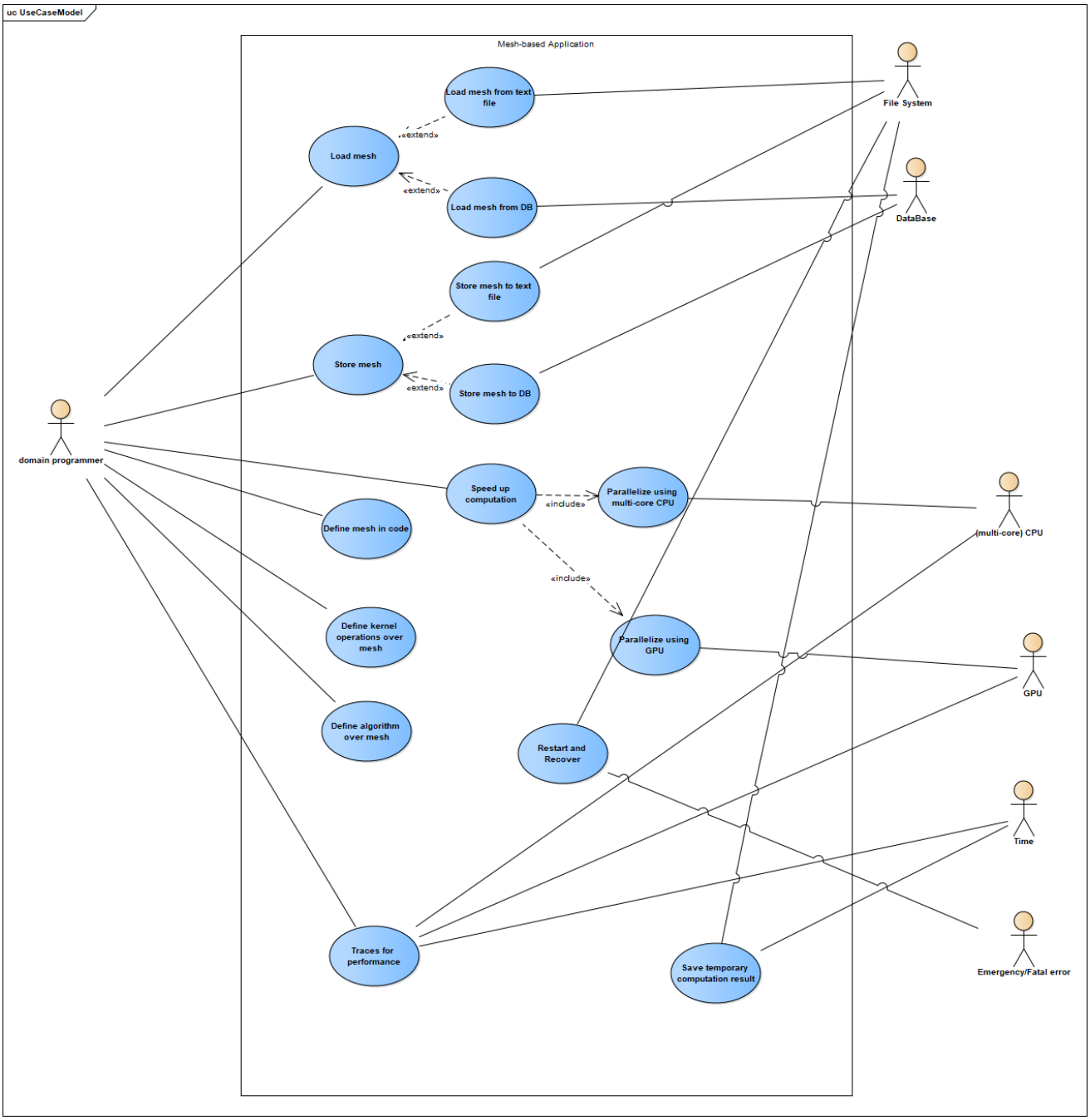
Behavioral Modeling

name: 陈灿辉 studentID: 17343008 class: 软工一班

Analyze the use cases we have developed for mesh-based applications, realize the use cases, turn these use cases into Interaction Diagrams. You may extend your previously developed requirements and you may also update your previously developed use case model for mesh based applications.

Use case Diagram

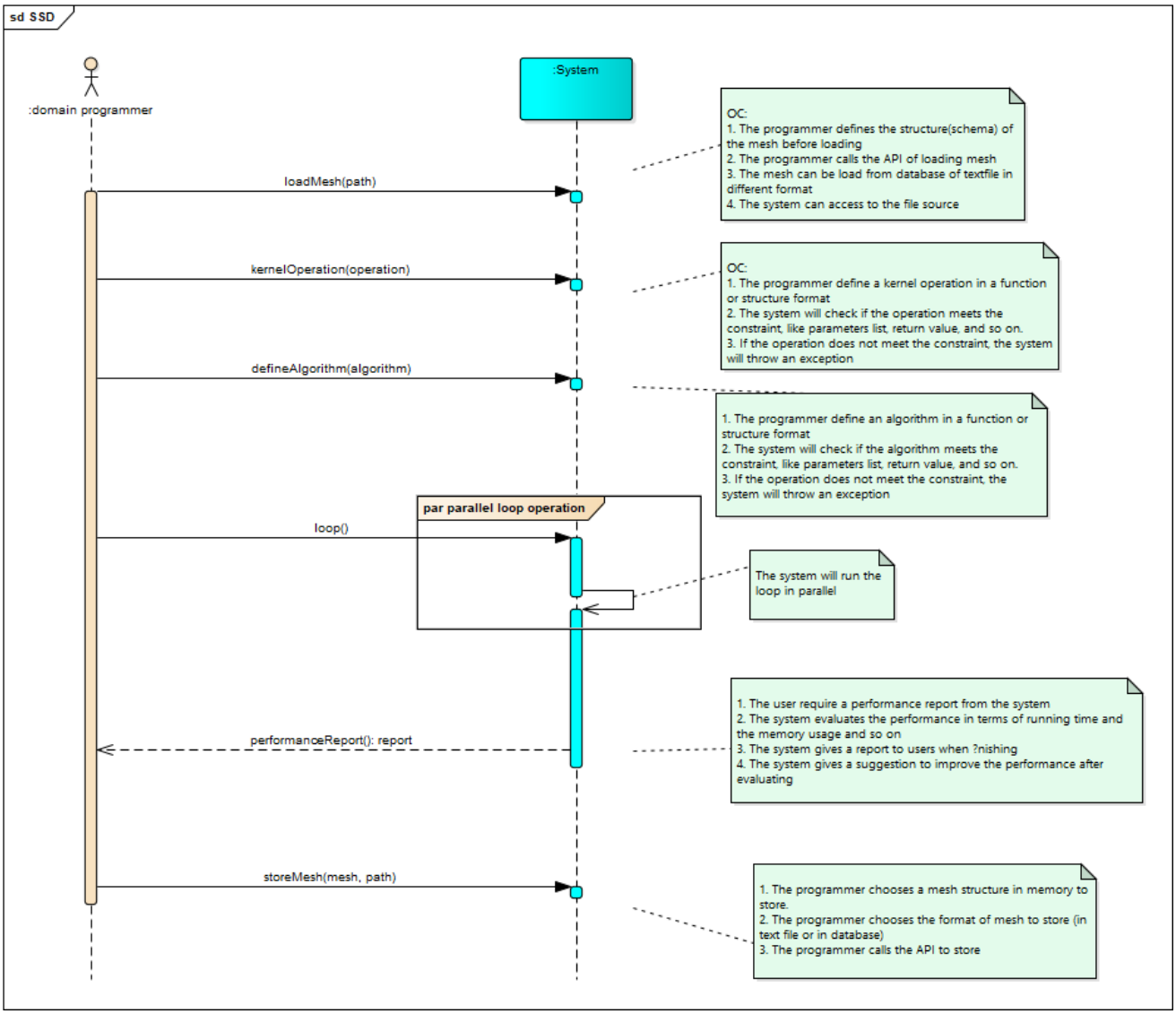
This is the use case diagram I develop in homework 3 for mesh-based application. The following behavioral modeling is based on the use case.



Interaction Diagram

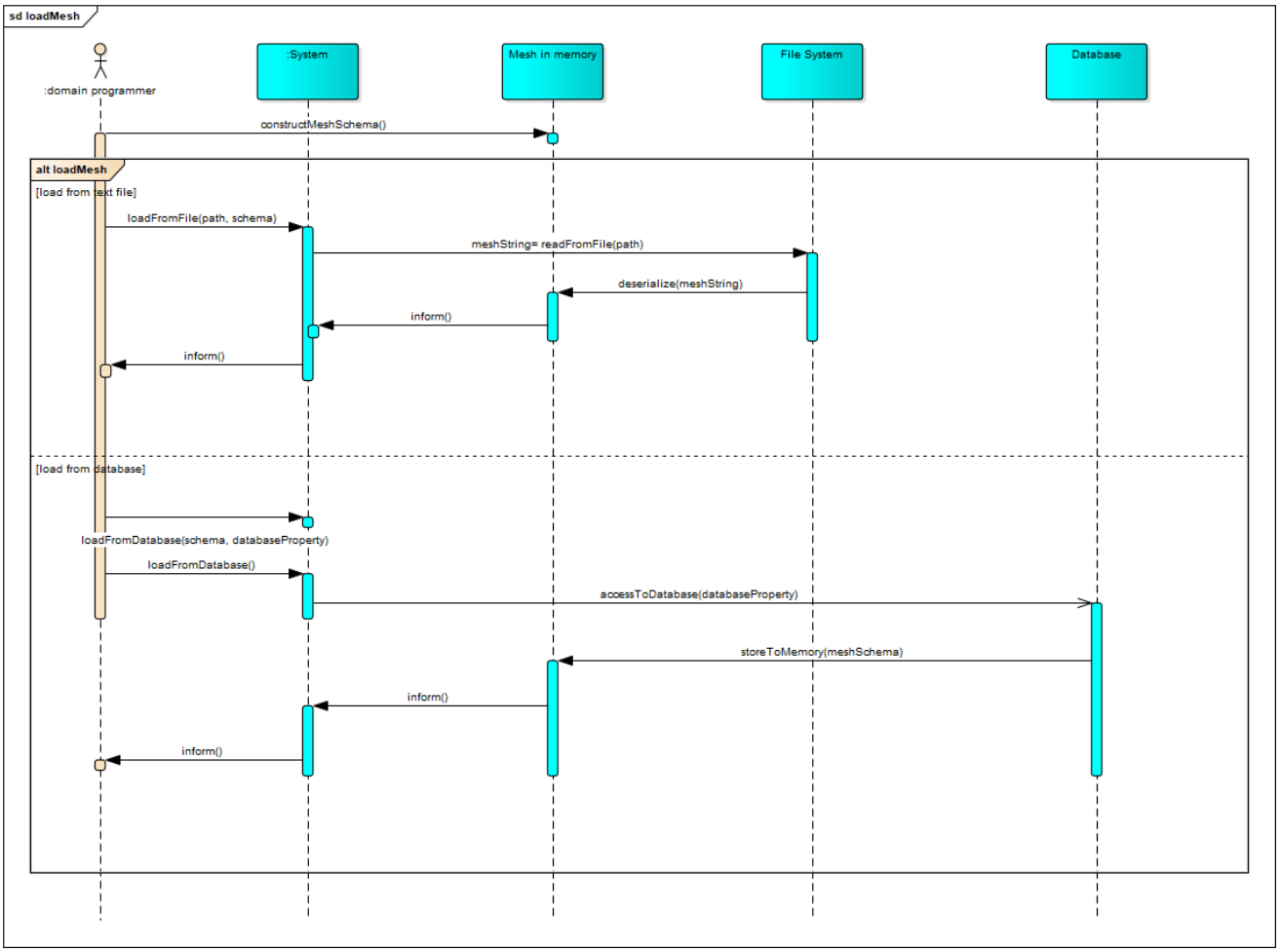
I will first develop a system sequence diagram (SSD) for the mesh-based application, and then I will develop the sequence diagram and coordination diagram for each detail use case.

System Sequence Diagram (SSD)

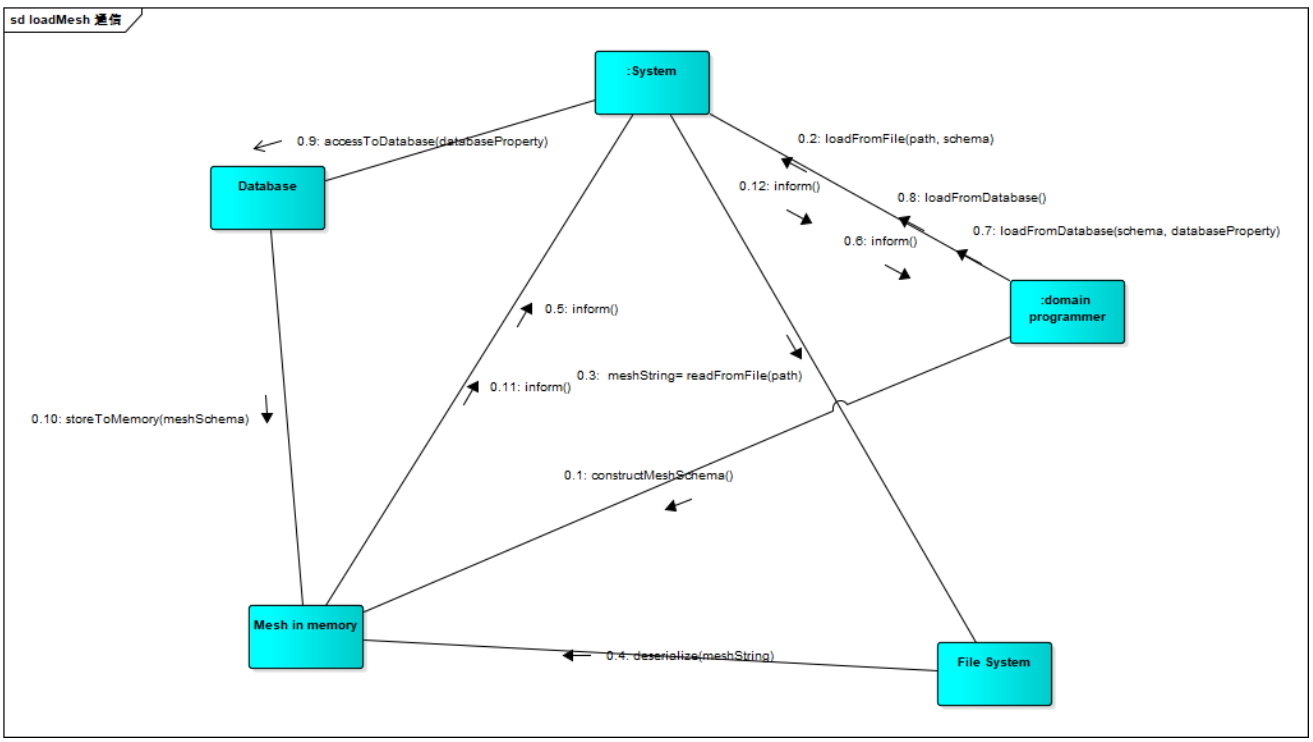


loadMesh

sequence diagram

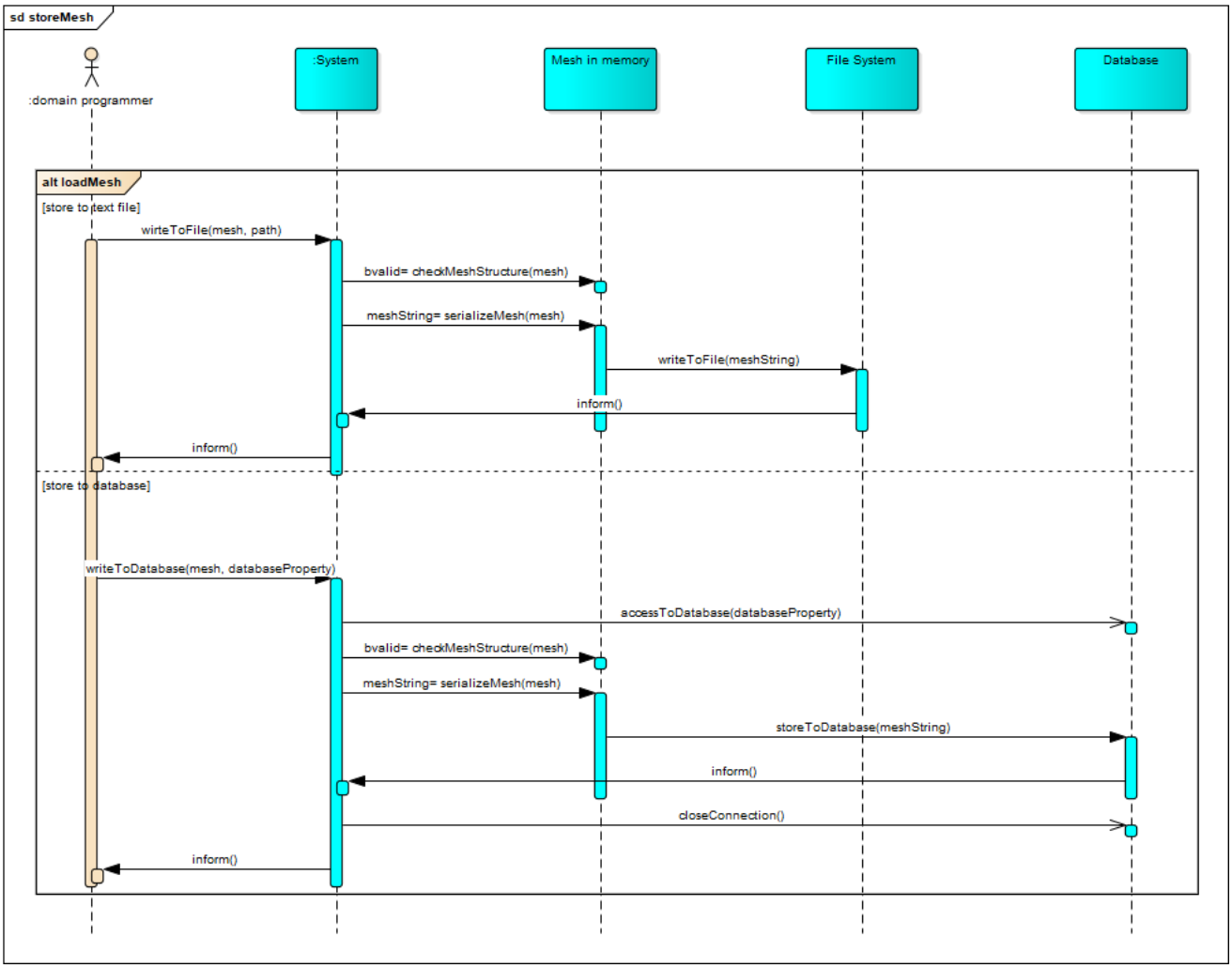


coordination diagram

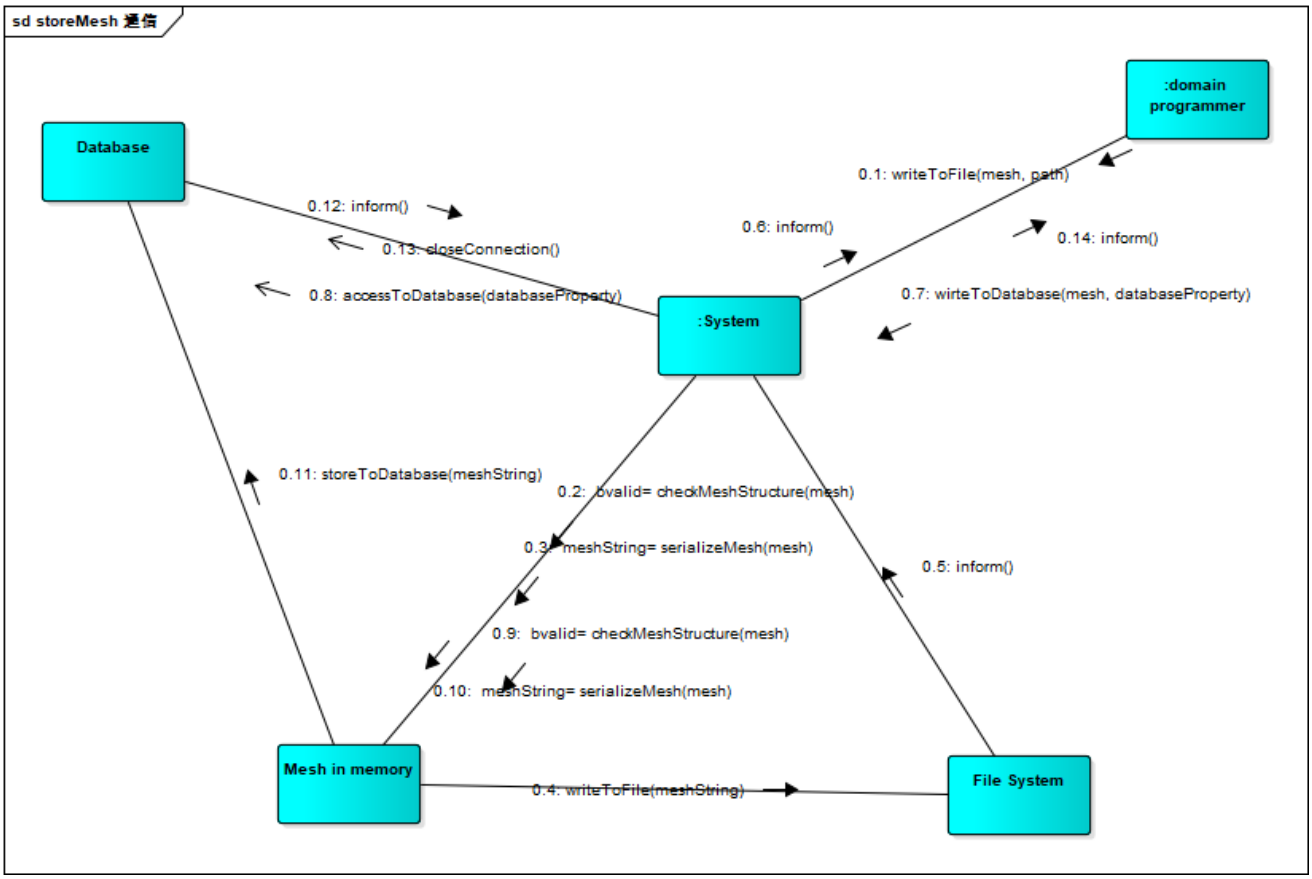


storeMesh

sequence diagram

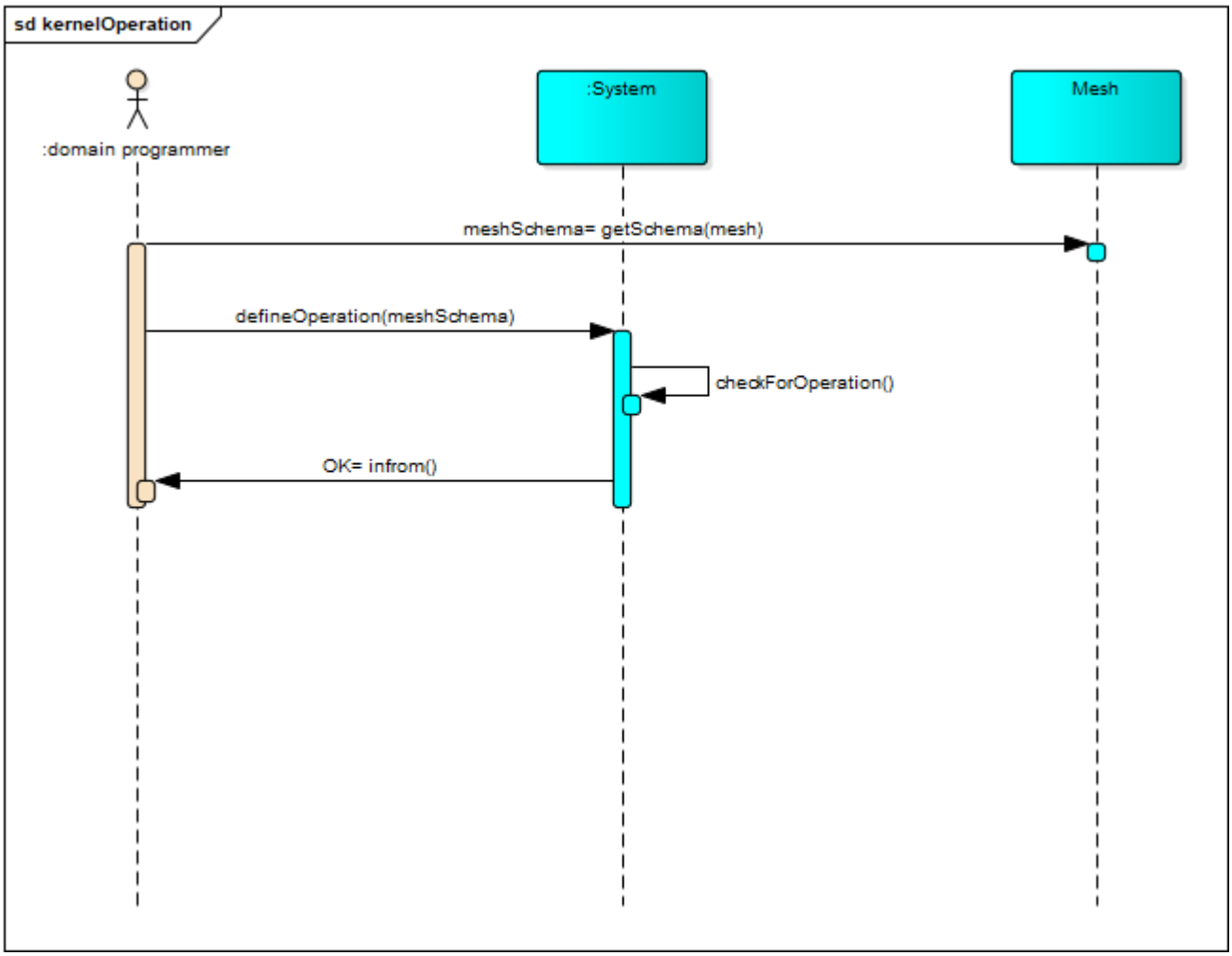


coordination diagram

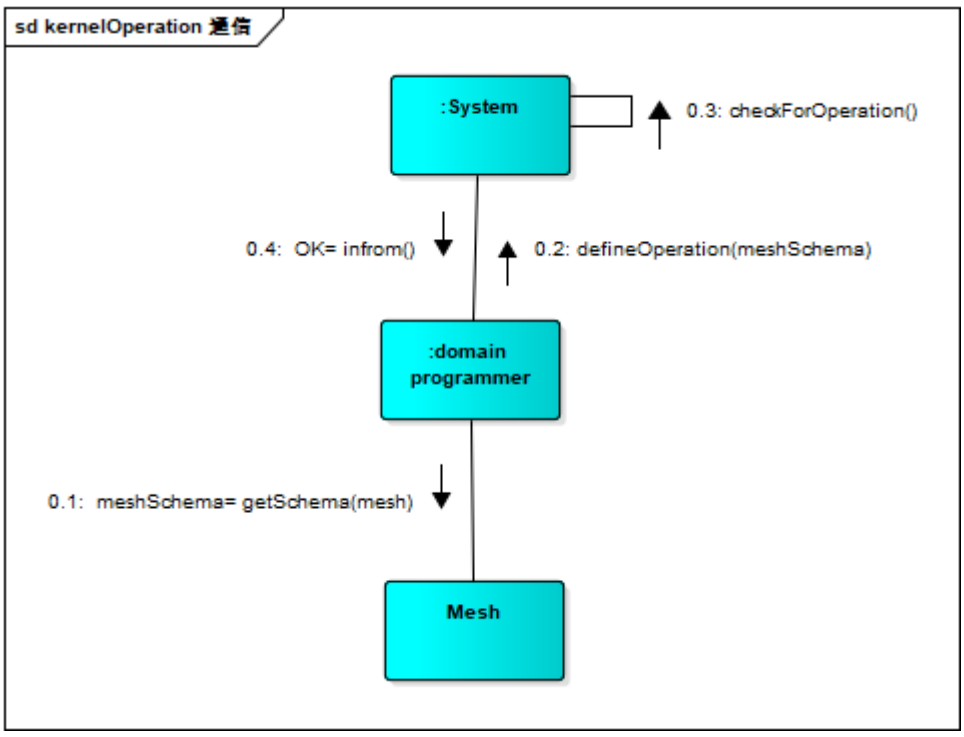


kernel Operation

sequence diagram

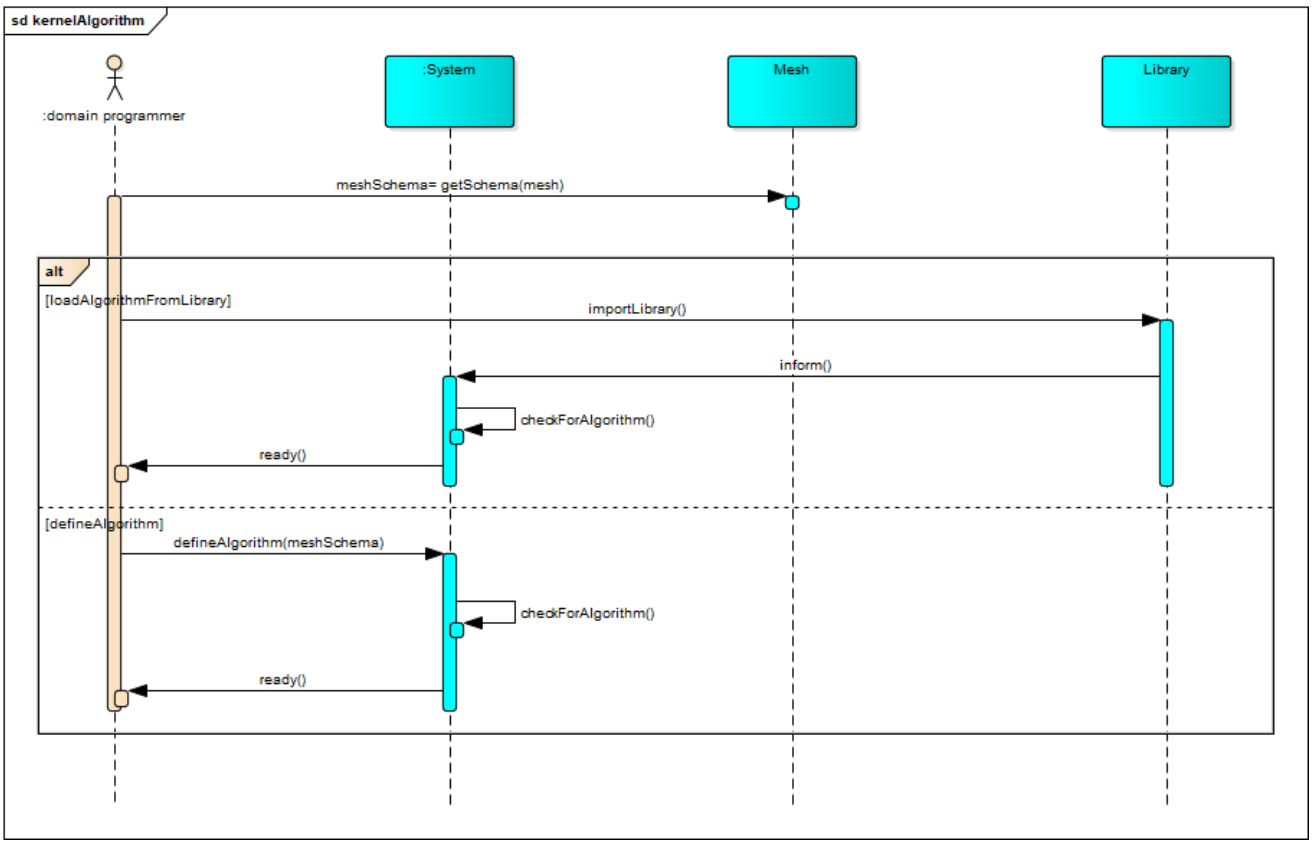


coordination diagram

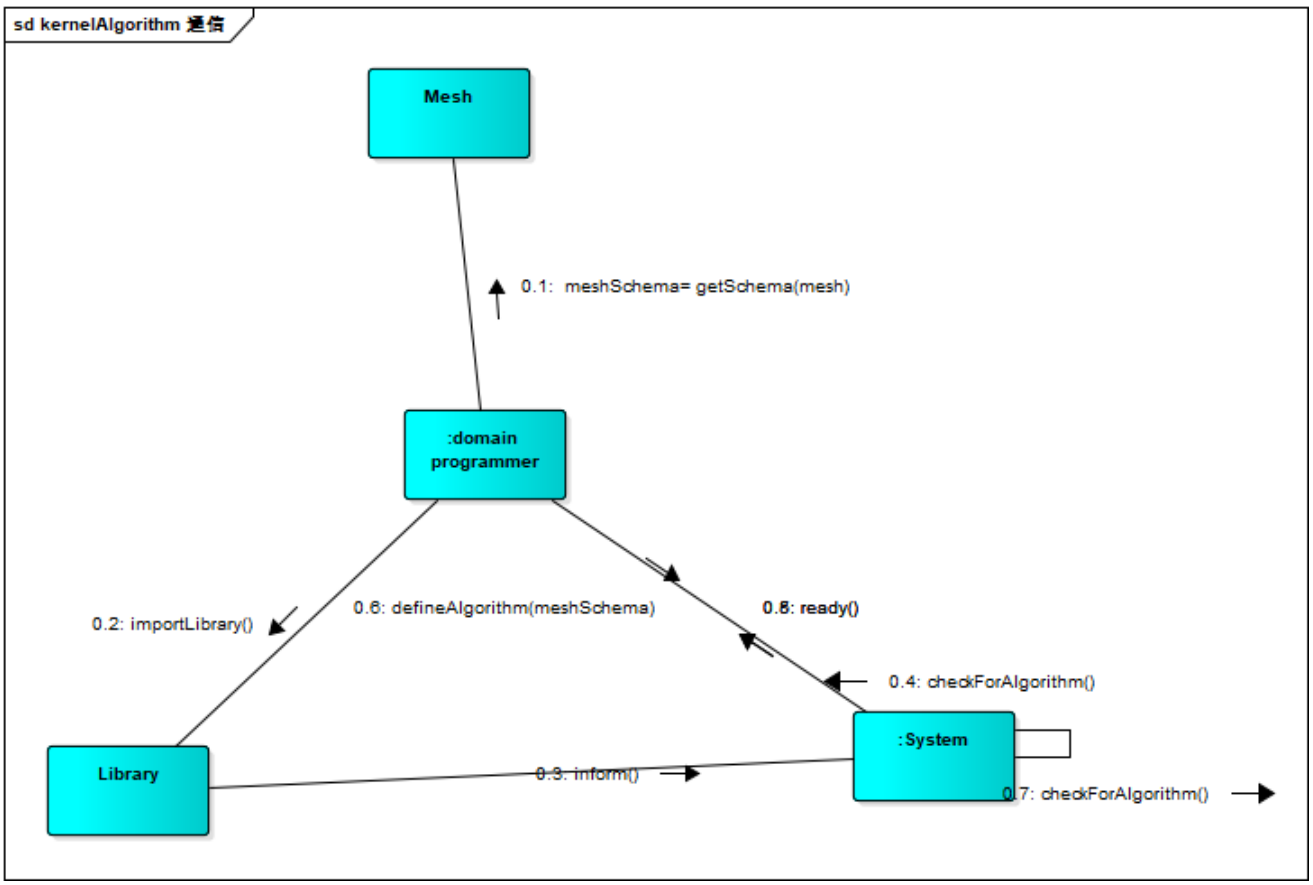


kernel Algorithm

sequence diagram

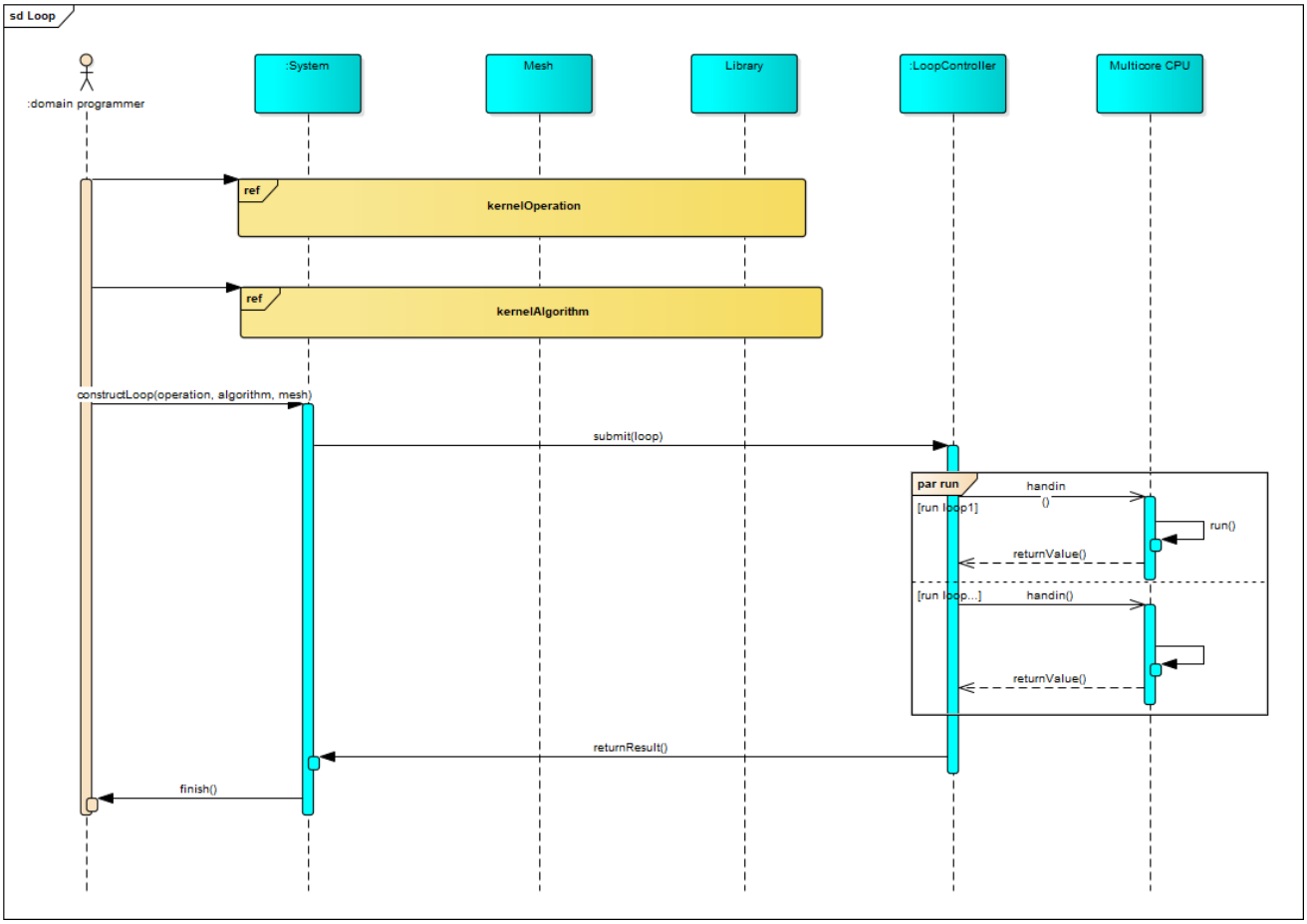


coordination diagram

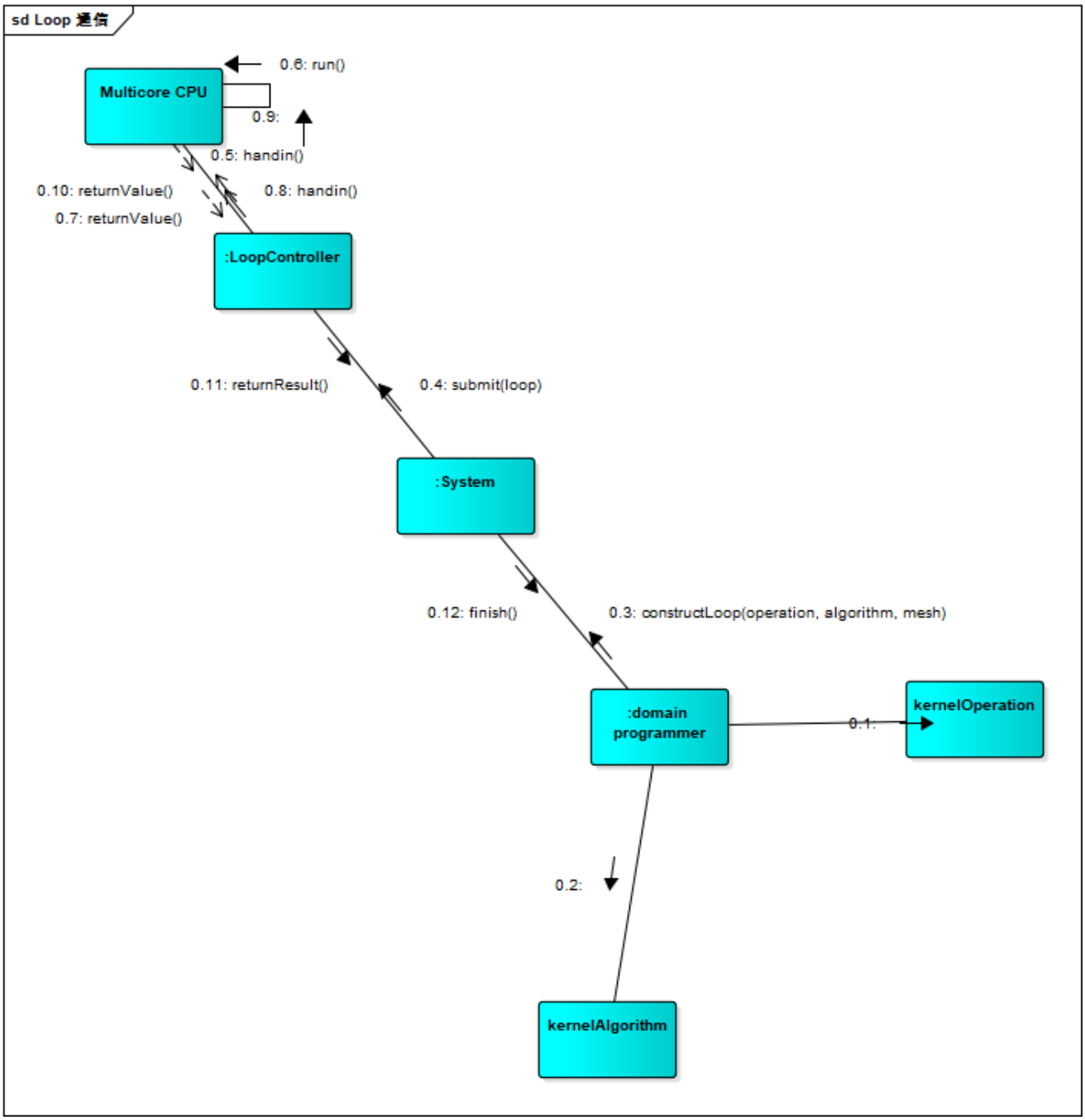


Loop

sequence diagram

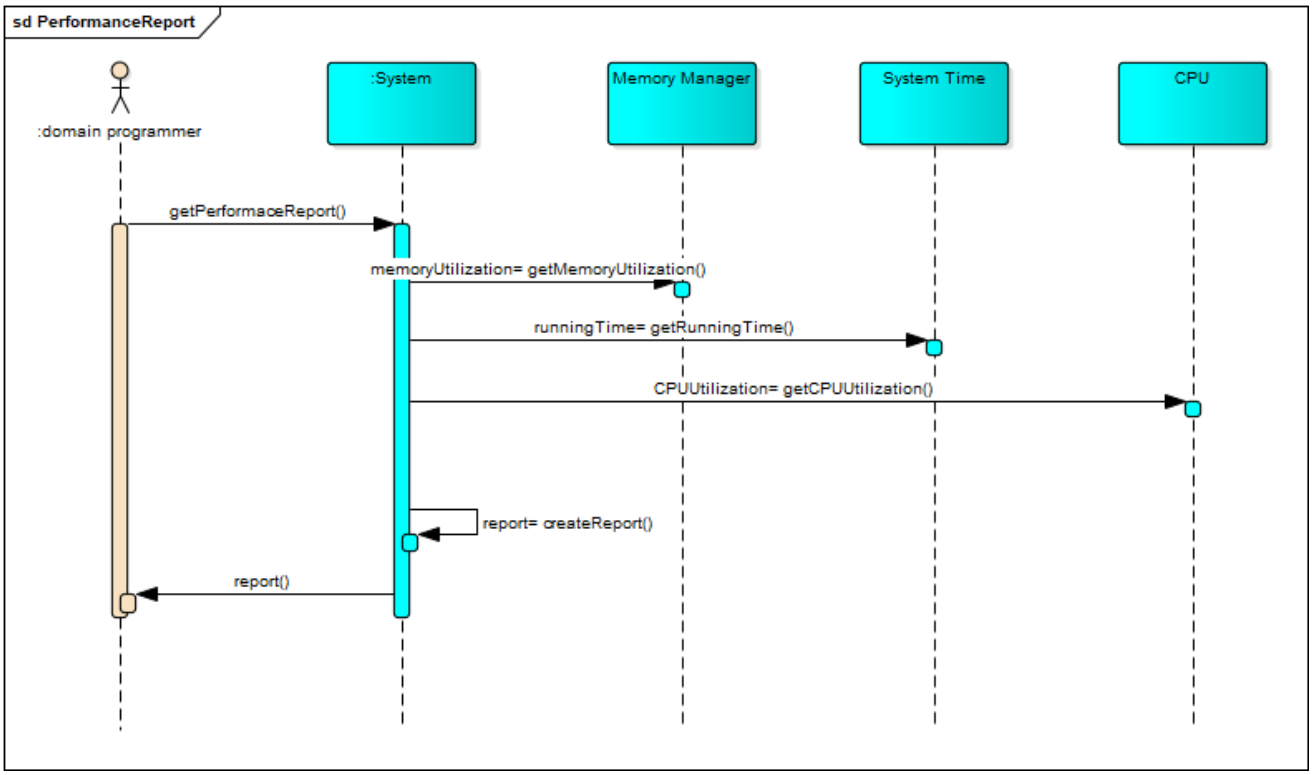


coordination diagram



Performance Report

sequence diagram



coordination diagram