Track B: Particle Methods – Part 2

PRACE Spring School 2012

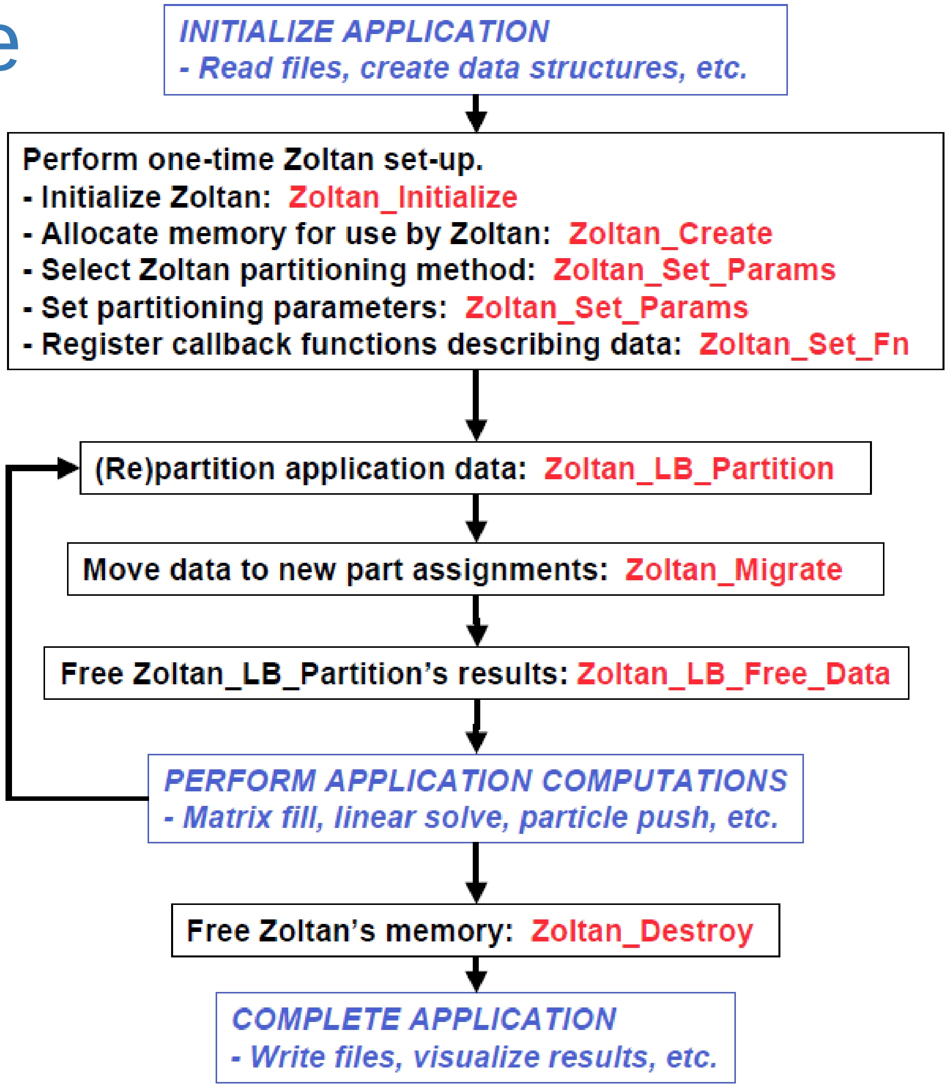
Maciej Cytowski (ICM UW)

PART2: Load-balancing and migration

- Zoltan set-up ~ 15 min.
- Load-balancing in Zoltan ~ 15 min.
- Hands-on exercises ~ 20 min.
- Migration in Zoltan ~ 15 min.
- Hands-on exercises ~ 25 min.

Typical Zoltan usage

- Use of Zoltan in a typical dynamic application.
- Calls to Zoltan functions are shown in red.
- Application operations are shown in blue.


INITIALIZE APPLICATION
- Read files, create data structures, etc.
Perform one-time Zoltan set-up.
- Initialize Zoltan: Zoltan_Initialize
- Allocate memory for use by Zoltan: Zoltan_Create
- Select Zoltan partitioning method: Zoltan_Set_Params
- Set partitioning parameters: Zoltan_Set_Params
- Register callback functions describing data: Zoltan_Set_Fn
(Re)partition application data: Zoltan_LB_Partition
Move data to new part assignments: Zoltan_Migrate
Free Zoltan_LB_Partition's results: Zoltan_LB_Free_Data
PERFORM APPLICATION COMPUTATIONS
- Matrix fill, linear solve, particle push, etc.
Free Zoltan's memory: Zoltan_Destroy
COMPLETE APPLICATION
- Write files, visualize results, etc.

Source:
„**Getting Started with Zoltan: a Short Tutorial**"
K.D. Devine, E.G. Boman, L.A. Riesen, U.V. Catalyurek, C. Chevalier

Zoltan set-up (decompose.c)

```
/* This function performs the initialization of Zoltan decomposition
        */
int decompositionInit(int argc,char **argv) {

  float version;

  Zoltan_Initialize(argc,argv,&version);
  if(rank==0) printf("Zoltan Version %.3f. Initialized.\n",version);

  ztn=Zoltan_Create(MPI_COMM_WORLD);

  Zoltan_Set_Param(ztn, "LB_METHOD","HSFC");
  Zoltan_Set_Param(ztn, "NUM_GID_ENTRIES", "1");
  Zoltan_Set_Param(ztn, "NUM_LID_ENTRIES", "1");
  Zoltan_Set_Param(ztn, "OBJ_WEIGHT_DIM", "1");
  Zoltan_Set_Param(ztn, "DEBUG_LEVEL","0");
  Zoltan_Set_Param(ztn, "KEEP_CUTS","1");
  Zoltan_Set_Param(ztn, "AUTO_MIGRATE", "1");

  Zoltan_Set_Fn(ztn, ZOLTAN_NUM_GEOM_FN_TYPE, (void
        (*)()) ztn_return_dimension, particles);
  Zoltan_Set_Fn(ztn, ZOLTAN_GEOM_FN_TYPE, (void (*)())
        ztn_return_coords, particles);
  Zoltan_Set_Fn(ztn, ZOLTAN_NUM_OBJ_FN_TYPE, (void (*)
        ()) ztn_return_num_node, particles);
  Zoltan_Set_Fn(ztn, ZOLTAN_OBJ_LIST_FN_TYPE, (void (*)
        ()) ztn_return_owned_nodes, particles);

  Zoltan_Set_Fn(ztn, ZOLTAN_OBJ_SIZE_FN_TYPE,(void (*)
        ()) ztn_return_particle_data_size, particles);
  Zoltan_Set_Fn(ztn, ZOLTAN_PACK_OBJ_FN_TYPE,(void (*)
        ()) ztn_pack, particles);
  Zoltan_Set_Fn(ztn, ZOLTAN_UNPACK_OBJ_FN_TYPE,(void
        (*)()) ztn_unpack, particles);
  Zoltan_Set_Fn(ztn, ZOLTAN_PRE_MIGRATE_PP_FN_TYPE,
        (void (*)()) ztn_pre, particles);
  Zoltan_Set_Fn(ztn, ZOLTAN_MID_MIGRATE_PP_FN_TYPE,
        (void (*)()) ztn_mid, particles);
  Zoltan_Set_Fn(ztn,
        ZOLTAN_POST_MIGRATE_PP_FN_TYPE,(void (*)())
        ztn_post, particles);
  return 0;
}
```

Zoltan Initialization

```
int Zoltan_Initialize (
      int argc,
      char **argv,
      float *ver);
```

- The **Zoltan_Initialize** function initializes MPI for Zoltan
- If the application uses MPI, this function should be called after calling MPI_Init
- If the application does not use MPI, this function calls MPI_Init for use by Zoltan
- This function is called with the argc and argv command-line arguments from the main program, which are used if **Zoltan_Initialize** calls MPI_Init
- Look at initialization in **decompose.c**

```
Zoltan_Initialize(argc,argv,&version);
```

Zoltan Create

```
struct Zoltan_Struct *Zoltan_Create (
      MPI_Comm communicator);
```

- The **Zoltan_Create** function allocates memory for storage of information to be used by Zoltan and sets the default values for the information
- The pointer returned by this function is passed to many subsequent functions

```
ztn=Zoltan_Create(MPI_COMM_WORLD);
```

Zoltan_Set_Param

```
int Zoltan_Set_Param (
      struct Zoltan_Struct *zz,
      char *param_name,
      char *new_val);
```

- **Zoltan_Set_Param** is used to alter the value of one of the parameters used by Zoltan
- All Zoltan parameters have reasonable default values, but this routine allows a user to provide alternative values if desired

- We will discuss each call of this function

Zoltan_Set_Param

- **Zoltan_Set_Param(ztn, "LB_METHOD","HSFC");**
  Load-balancing method is set to Hilbert Space Filling Curve algorithm.

- **Zoltan_Set_Param(ztn, "NUM_GID_ENTRIES", "1");**
  Global ID of particles is based on a single value (rather than on a vector).

- **Zoltan_Set_Param(ztn, "NUM_LID_ENTRIES", "1");**
  Local ID of particles is based on a single value (rather than on a vector).

- **Zoltan_Set_Param(ztn, "OBJ_WEIGHT_DIM", "1");**
  This parameters enables usage of object weights. This is a very nice feature of Zoltan partitioning.

Zoltan_Set_Param

- **Zoltan_Set_Param(ztn, "DEBUG_LEVEL","0");**

  This sets Zoltan to quiet mode. No output will be given unless an error or warning is produced. Try to set to 1,2,... .

- **Zoltan_Set_Param(ztn, "KEEP_CUTS","1");**

  The information about domain cuts and bounding boxes will be kept. This parameter enables usage of box and point assignment functions.

- **Zoltan_Set_Param(ztn, "AUTO_MIGRATE", "1");**

  We want data migration to be performed by Zoltan automatically. Some additional Zoltan query functions have to be defined.

Zoltan_Set_Fn

```
int Zoltan_Set_Fn (
        struct Zoltan_Struct *zz,
        ZOLTAN_FN_TYPE fn_type,
        void (*fn_ptr)(),
        void *data);
```

- **Zoltan_Set_Fn** registers an application-supplied query function in the Zoltan structure
- All types of query functions can be registered through calls to **Zoltan_Set_Fn**
- **Query function = callback function**
- **„Don't call us, we'll call you"** – programmer defines and registers library functions rather than using them. The library uses registered functions when needed.

- We will discuss each call of this function

Zoltan_Set_Fn – load-balancing specific

Different query functions are required to be defined for different Zoltan LB methods. For HSFC we need to define:

•**Zoltan_Set_Fn(ztn, ZOLTAN_NUM_GEOM_FN_TYPE, (void (*)()) ztn_return_dimension, particles);**

A **ZOLTAN_NUM_GEOM_FN** query function returns the number of values needed to express the geometry of an object. For example, for a two-dimensional mesh-based application, (x,y) coordinates are needed to describe an object's geometry; thus the **ZOLTAN_NUM_GEOM_FN** query function should return the value of two. For a similar three-dimensional application, the return value should be three.

•**Zoltan_Set_Fn(ztn, ZOLTAN_GEOM_FN_TYPE, (void (*)()) ztn_return_coords, particles);**

A **ZOLTAN_GEOM_FN** query function returns a vector of geometry values for a given object. The geometry vector is allocated by Zoltan to be of the size returned by a **ZOLTAN_NUM_GEOM_FN** query function.

•**Zoltan_Set_Fn(ztn, ZOLTAN_NUM_OBJ_FN_TYPE, (void (*)()) ztn_return_num_node, particles);**

A **ZOLTAN_NUM_OBJ_FN** query function returns the number of objects that are currently assigned to the processor.

Zoltan_Set_Fn – load-balancing specific

Different query functions are required to be defined for different Zoltan LB methods. For HSFC we need to define:

•**Zoltan_Set_Fn(ztn, ZOLTAN_OBJ_LIST_FN_TYPE, (void (*)()) ztn_return_owned_nodes, particles);**

A **ZOLTAN_OBJ_LIST_FN** query function fills two (three if weights are used) arrays with information about the objects currently assigned to the processor. Both arrays are allocated (and subsequently freed) by Zoltan; their size is determined by a call to a **ZOLTAN_NUM_OBJ_FN** query function to get the array size.

This function is already defined:

```
            void ztn_return_owned_nodes(void *data, int num_gid_entries, int num_lid_entries,
ZOLTAN_ID_PTR global_ids, ZOLTAN_ID_PTR local_ids, int wgt_dim, float *obj_wgts, int *ierr) {
                        int i;
                        struct particle_data *p = (struct particle_data*) data;
                        for(i=0;i<lnp;i++) {
                        global_ids[i*num_gid_entries]=p[i].gid;
                                    local_ids[i*num_lid_entries]=i;
                        obj_wgts[i]=1.0;
                        }
            }
```

Hands-on load-balancing in Zoltan – Exercise 2

- Write the content of three Zoltan load-balancing specific query functions:
  - **ztn_return_dimension**
  - **ztn_return_coords**
  - **ztn_return_num_node**

- Uncomment the **decompositionInit()** call in **main.c**
- Compile & run the code

Zoltan_Set_Fn – migration specific

- **Zoltan_Set_Fn(ztn, ZOLTAN_OBJ_SIZE_FN_TYPE,(void (*)()) ztn_return_particle_data_size, particles);**

  A **ZOLTAN_OBJ_SIZE_FN** query function returns the size (in bytes) of the data buffer that is needed to pack all of a single object's data. We assume that we send the whole particle_data between processes.

- **Zoltan_Set_Fn(ztn, ZOLTAN_PACK_OBJ_FN_TYPE,(void (*)()) ztn_pack, particles);**

  A **ZOLTAN_PACK_OBJ_FN** query function allows the application to tell Zoltan how to copy all needed data for a given object into a communication buffer. The object's data can then be sent to another processor as part of data migration. It may also perform other operations, such as removing the object from the processor's data structure. This routine is called by **Zoltan_Migrate** for each object to be sent to another processor.

- **Zoltan_Set_Fn(ztn, ZOLTAN_UNPACK_OBJ_FN_TYPE,(void (*)()) ztn_unpack, particles);**

  A **ZOLTAN_UNPACK_OBJ_FN** query function allows the application to tell Zoltan how to copy all needed data for a given object from a communication buffer into the application's data structure. This operation is needed as the final step of importing objects during data migration. The query function may also perform other computation, such as building request lists for related data. This routine is called by **Zoltan_Migrate** for each object to be received by the processor.

Zoltan_Set_Fn – migration specific

- **Zoltan_Set_Fn(ztn, ZOLTAN_PRE_MIGRATE_PP_FN_TYPE,(void (*)()) ztn_pre, particles);**
  A **ZOLTAN_PRE_MIGRATE_PP_FN** query function performs any pre-processing desired by the application. If it is registered, it is called at the beginning of the **Zoltan_Migrate** routine.

- **Zoltan_Set_Fn(ztn, ZOLTAN_MID_MIGRATE_PP_FN_TYPE,(void (*)()) ztn_mid, particles);**
  A **ZOLTAN_MID_MIGRATE_PP_FN** query function performs any processing desired by the application between the packing and unpacking of objects being migrated. If it is registered, it is called after export objects are packed in **Zoltan_Migrate**;

```
void ztn_mid(…) {
  int pos,i;
  struct particle_data *p = (struct particle_data*) data;
  pos=0;
  for(i=0;i<lnp;i++) {
    if(i!=pos && p[i].gid!=-1) { p[pos]=p[i]; p[pos]=p[i]; }
    if(p[i].gid!=-1) pos++;
  }
  lnp=lnp-num_export;
}
```

- **Zoltan_Set_Fn(ztn, ZOLTAN_POST_MIGRATE_PP_FN_TYPE,(void (*)()) ztn_post, particles);**
  A **ZOLTAN_POST_MIGRATE_PP_FN** query function performs any post-processing desired by the application. If it is registered, it is called at the end of the **Zoltan_Migrate** routine.

Load-balancing in Zoltan

```
/* This function performes the decomposition */
int decompose() {

  int rc;
  int i;

  rc = Zoltan_LB_Partition(ztn, /* input (all remaining fields are output) */
        &changes,              /* 1 if partitioning was changed, 0 otherwise */
        &numGidEntries,        /* Number of integers used for a global ID */
        &numLidEntries,        /* Number of integers used for a local ID */
        &numImport,            /* Number of objects to be sent to me */
        &importGlobalGids,     /* Global IDs of objects to be sent to me */
        &importLocalGids,      /* Local IDs of objects to be sent to me */
        &importProcs,          /* Process rank for source of each incoming object */
        &importToPart,         /* New partition for each incoming object */
        &numExport,            /* Number of objects I must send to other processes*/
        &exportGlobalGids,     /* Global IDs of the objects I must send */
        &exportLocalGids,      /* Local IDs of the objects I must send */
        &exportProcs,          /* Process to which I send each of the objects */
        &exportToPart);        /* Partition to which each object will belong */

  if (rc != ZOLTAN_OK)
    printf("Error in Zoltan library\n");

  // Free the arrays allocated by Zoltan_LB_Partiotion
  Zoltan_LB_Free_Part(&importGlobalGids,
        &importLocalGids,&importProcs, &importToPart);
  Zoltan_LB_Free_Part(&exportGlobalGids,
        &exportLocalGids,&exportProcs, &exportToPart);

  return 0;

}
```

Clear the partitioning temporary arrays

Perform the partitioning

Hands-on data migration in Zoltan – Exercise 3

- Write the content of three Zoltan migration specific query functions:
  - **ztn_return_particle_data_size**
  - **ztn_pack** (remember to mark exported particles as: `p[i].gid=-1`)
  - **ztn_unpack**
  - **ztn_pre** (print statistics of migration – number of exported and imported particles for each process)

- Compile & run the code
- Uncomment the **decompose()** call in **main.c**
- Increase the number of iterations to 2 (in the **partilces.ll** file)
- Copy two resulting files to your laptop
- Use VisNow for visualization of load-balancing
- **Desired result**: colors indicating distinct processes' domains

Hint 1

```
void ztn_pack(void *data,int num_gid_entries,int
    num_lid_entries,ZOLTAN_ID_PTR global_id,ZOLTAN_ID_PTR local_id,int
    dest,int size,char *buf,int *ierr) {
    struct particle_data *p = (struct particle_data*) data;
    memcpy(buf,&(p[(int)(*local_id)]),sizeof(struct particle_data));
     p[(int)(*local_id)].gid=-1; // Mark local particle as exported
}
```