ParaGrapher API Documentation

May 4, 2024

ParaGrapher [1] is a graph loading API and library designed to provide efficient loading and decompression of the large-scale graph datasets. The design steps, functionality, and evaluation of the ParaGrapher has been explained in a separate document [Koohi Esfahani et al.(2024)Koohi Esfahani, D'Antonio, Tauhidi, Mai, and Va and in this document, we explain the functions and definitions in ParaGrapher API. The ParaGrapher header file can be accessed on https://github.com/DIPSA-QUB/ParaGrapher/blob/main/include/paragrapher.h.

Please note that most functions have two arguments `void**` **args** and `int` **argc** that have been used to pass additional arguments (for sending input to the library or receiving data from library) that may be required for particular graph types.

1 ParaGrapher Initialization

This function initializes the ParaGrapher library. This function must be called before using any other library functions. Upon calling this function, ParaGrapher iterates over its inner files that have implemented the API for loading different graph formats and creates a list of their functions which is used for accessing graphs.

```
int paragrapher_init()
```

Returns:

- `0` on success; `-1` on failure

2 Opening A Graph

This function opens a graph specified by type and filename.

```
paragrapher_graph* paragrapher_open_graph(
   char* name, paragrapher_graph_type type,
   void** args, int argc)
```

Parameters:

- `name`: The file name or identifier for the graph.
- `type`: The type of the graph, as defined by `paragrapher_graph_type` enum and listed in Table 1.
- `args`: Additional arguments required for opening the graph, specific to the graph type.
- `argc`: The number of additional arguments.

Returns:

- A pointer to a `paragrapher_graph` structure if successful.
- `NULL` if the graph could not be opened due to an invalid type or other errors.

[1] https://blogs.qub.ac.uk/DIPSA/ParaGrapher/

Type	Vertex ID Size (Bytes)	Vertex Weight Size (Bytes)	Edge Weight Size (Bytes)
CSX_WG_400_AP	4	0	0
CSX_WG_800_AP	8	0	0
CSX_WG_404_AP	4	0	4

Table 1: Options for paragrapher_graph_type

3 Getting Info and Setting Configuration

This function configures the settings or retrieves configurations.

```
int paragrapher_get_set_options(
   paragrapher_graph* graph,
   paragrapher_request_type request_type,
   void** args, int argc)
```

Parameters:

- graph: A pointer to the graph to configure.
- request_type: The type of request specifying the option to get or set given by paragrapher_request_type enum and listed in Table 2.
- args: Arguments for the request.
- argc: The number of arguments in args.

Returns:

- 0 on success.
- A negative integer, on error, with the specific value indicating the type of error.

Option	Type (argv[0])	Type (argv[1])
PARAGRAPHER_REQUEST_GET_GRAPH_PATH Returns the file path of the graph	char[PATH_MAX]	-
PARAGRAPHER_REQUEST_GET_VERTICES_COUNT Returns the number of vertices in the graph	unsigned long	-
PARAGRAPHER_REQUEST_GET_EDGES_COUNT Returns the number of edges in the graph	unsigned long	-
PARAGRAPHER_REQUEST_LIB_USES_OWN_BUFFERS Checks if the library uses its own buffers	unsigned long	-
PARAGRAPHER_REQUEST_LIB_USES_USER_ARRAYS Checks if the library uses arrays provided by the user	unsigned long	-
PARAGRAPHER_REQUEST_SET_BUFFER_SIZE Sets the size of the buffer used by the library	unsigned long	-
PARAGRAPHER_REQUEST_GET_BUFFER_SIZE Returns the current buffer size used by the library	unsigned long	-
PARAGRAPHER_REQUEST_SET_MAX_BUFFERS_COUNT Sets the maximum number of buffers the library can use	unsigned long	-
PARAGRAPHER_REQUEST_GET_MAX_BUFFERS_COUNT Returns the maximum number of buffers the library can use	unsigned long	-
PARAGRAPHER_REQUEST_READ_STATUS Queries the status of a read operation	paragrapher_read_request*	unsigned long
PARAGRAPHER_REQUEST_READ_TOTAL_CALLBACKS Returns the total number of callbacks triggered during reading	paragrapher_read_request*	unsigned long
PARAGRAPHER_REQUEST_READ_EDGES Initiates reading of edge data	paragrapher_read_request*	unsigned long

Table 2: paragrapher_request_type options and return types

4 Accessing CSX Offsets and Weights

4.1 Getting offsets and weights

The following functions return the offsets or weights of vertices in a CSX graph within a specified range.

```
void* paragrapher_csx_get_offsets(
   paragrapher_graph* graph,
   void* offsets,
   unsigned long start_vertex,
   unsigned long end_vertex,
   void** args, int argc)

void* paragrapher_csx_get_vertex_weights(
   paragrapher_graph* graph,
   void* weights,
   unsigned long start_vertex,
   unsigned long end_vertex,
   void** args, int argc)
```

Parameters:

- `graph`: A pointer to the graph.
- `offsets`, `weights`,: A buffer to store the offsets or weights.
- `start_vertex`, `end_vertex`: The range of vertices to process.
- `args`, `argc`: Additional arguments for the operation.

Returns:

- A pointer to the buffer containing the offsets if successful.
- NULL on failure.

4.2 Releasing offsets and weights buffers

The following function is used to release the offsets or weights array returned by the library.

```
void paragrapher_csx_release_offsets_weights_arrays(
   paragrapher_graph* graph, void* array)
```

Parameters:

- `graph`: A pointer to the graph.
- `array`: The array to be released.

5 Accessing CSX Edges

5.1 Callback function

The callback function for handling data received from asynchronous subgraph reading operations in CSX graph formats is defined as follows. When a block of the edges is read by the library, this callback is invoked to inform user about the loaded data. Depending on the size of buffer and the number of edges, the callback function may be called multiple times. The callback functions is defined by the user and invoked by the library

```
typedef void (*paragrapher_csx_callback)(
   paragrapher_read_request* request,
   paragrapher_edge_block* eb,
   void* offsets,
   void* edges,
   void* buffer_id,
```

```
   void* args);
```

Parameters:

- `request`: A pointer to the `paragrapher_read_request` structure.
- `eb`: A pointer to the structure describing the block of edges have been loaded.
- `offsets`: A pointer to an array containing the vertex offsets.
- `edges`: A pointer to an array containing the edges within the specified block.
- `buffer_id`: A pointer to an identifier for the buffer used during the operation.
- `callback_args`: A pointer to a user-defined parameter that is passed to the `csx_get_subgraph` by the user and the library redirects it to the callback function.

`paragrapher_edge_block` **structure**:

```
typedef struct {
    unsigned long start_vertex; // Start vertex index of the edge block
    unsigned long start_edge;   // Start edge index associated with the start vertex
    unsigned long end_vertex;   // End vertex index of the edge block
    unsigned long end_edge;     // End edge index associated with the end vertex
} paragrapher_edge_block;
```

5.2 Loading/Decompressing Edges

The following function starts loading a CSX graph or its subgraph (specified by `eb`). Depending on the implementation, load can be done synchronously (i.e., the `edges` array is filled by the library) or asynchronously (i.e., the `callback` function is called multiple times to pass read block of edges to user).

```
paragrapher_read_request* csx_get_subgraph(
   paragrapher_graph* graph,
   paragrapher_edge_block* eb,
   void* offsets, void* edges,
   paragrapher_csx_callback callback_args,
   void* callback_args, void** args, int argc);
```

Parameters:

- `graph`: A pointer to the graph structure where the subgraph is to be extracted.
- `eb`: A pointer to the `paragrapher_edge_block` structure that specifies the range of vertices and edges for which the subgraph should be extracted. This includes start and end vertices and edges.
- `offsets`: A pointer to an array where the offsets of the vertices will be stored. This parameter is used when the graph is loaded synchronously and the the library fills the pre-allocated arrays by the user.
- `edges`: A pointer to an array where the edges of the subgraph will be stored. This parameter is used when the graph is loaded synchronously and the the library fills the pre-allocated arrays by the user.
- `callback`: The callback function that the library calls to pass blocks of edges to the user. It must conform to the `paragrapher_csx_callback` signature. This parameter is used when the graph is loaded asynchronously and the the library passes its buffers to the user to access edges.
- `callback_args`: A pointer to any user-defined data that should be passed to the callback function.
- `args`: Additional format-specific arguments provided as an array of void pointers.
- `argc`: The count of additional arguments provided in args.

Returns:

- A pointer to a `paragrapher_read_request` structure if successful
- NULL on failure

5.3 Release Buffers

The following function should be used at the end of `callback` function to inform the library that the buffer will not be used any further and its memory can be reused by the library.

```
void csx_release_read_buffers(
   paragrapher_read_request* request,
   paragrapher_edge_block* eb,
   void* buffer_id);
```

Parameters:

- `request`: A pointer to the `paragrapher_read_request` returned by `paragrapher_csx_get_subgraph`.
- `eb`: A pointer to the `paragrapher_edge_block` structure indicating the specific range of edges and vertices involved in the request.
- `buffer_id`: A pointer to the identifier for the buffer used during the read operation. This identifier is typically provided during the callback execution and is used to manage and release the correct buffer.

5.4 Release Reader

The following function releases all resources associated with a returned `paragrapher_read_request` and should be called upon completion of the load process.

```
void csx_release_read_request(
   paragrapher_read_request* request);
```

Parameters:

- `request`: A pointer to the `paragrapher_read_request` structure that represents an ongoing or completed read request.

6 Accessing COO Edges

This function is similar to `csx_get_subgraph()` but for loading a COO graph or its subgraph synchronously or asynchronously.

```
paragrapher_read_request* coo_get_edges(
   paragrapher_graph* graph,
   unsigned long start_row,
   unsigned long end_row,
   void* edges,
   paragrapher_coo_callback callback,
   void* callback_args, void** args, int argc);
```

Parameters:

- `graph`: A pointer to the graph structure from which edges are to be read. This graph should be formatted in COO (Coordinate list) format.
- `start_row`: The starting row index in the edge list from where to begin reading.
- `end_row`: The ending row index in the edge list up to which edges should be read. If set to -1UL, it indicates that reading should continue until the end of the edge list.
- `edges`: A pointer to an array where the edges between `start_row` and `end_row` will be stored in the synchronous call.
- `callback`: A callback function invoked by the library when implementing an asynchronous load. The callback function is used to handle the edges read from the graph. This function should conform to the `paragrapher_coo_callback` signature.
- `callback_args`: A pointer to any user-defined data that should be passed to the callback function.
- `args`: Additional reader-specific arguments provided as an array of void pointers.

- `argc`: The count of additional arguments provided in args.

Returns:

- Pointer to a `paragrapher_read_request` if the operation is initiated successfully.
- NULL on failure.

7 Releasing Graph

The following function releases the resources allocated by the library for accessing a graph and should be called as the last step of accessing/loading a graph.

```
int paragrapher_release_graph(paragrapher_graph* graph, void** args, int argc)
```

Parameters:

- `graph`: A pointer to the graph to be released.
- `args`: Additional arguments required for releasing the graph, specific to the graph type.
- `argc`: The number of additional arguments.

Returns:

- `0` on success; `-1` on failure.

References

[1] Mohsen Koohi Esfahani, Marco D'Antonio, Syed Ibtisam Tauhidi, Thai Son Mai, and Hans Vandierendonck. Selective parallel loading of large-scale compressed graphs with paragrapher. 2024. URL: https://doi.org/10.48550/arXiv.2404.19735, arXiv:2404.19735.