NEMESIS I: A Set of Functions for Describing Unstructured Finite-Element Data on Parallel Computers

Gary L. Hennigan and John N. Shadid

December 2, 1998

Abstract

NEMESIS I is an enhancement to the EXODUS II finite element database model used to store and retrieve data for unstructured parallel finite element analyses. NEMESIS I adds data structures which facilitate the partitioning of a scalar (standard serial) EXODUS II file onto parallel disk systems found on many parallel computers. Since the NEMESIS I application programming interface (API) can be used to append information to an existing EXODUS II database, any existing software that reads EXODUS II files can be used on files which contain NEMESIS I information. The NEMESIS I information is written and read via C or C++ callable functions which compromise the NEMESIS I API.

Acknowledgments

The authors would like to thank the other members of the Salsa team who's suggestions and assistance were invaluable. Those members are Scott Hutchinson (SNL 9221), Harry Moffat (SNL 1126), Karen Devine (SNL 9224) and Andrew Salinger (SNL 9221). Equally important were the contributions from James Peery (SNL 9231) and Stephan Attaway (SNL 9118). Also, much of the code was based heavily on the EXODUS II API written by Larry Schoof (SNL 9225), Vic Yarberry (SNL 9225) and James Schutt (SNL 9111).

This work was partially supported under the Joint DoD/DOE Munitions Technology Development Program, and sponsored by the Office of Munitions of the Secretary of Defense.

Contents

1 Introduction

NEMESIS I is a set of C or C++ callable functions used to add information about parallel partitioning to an EXODUS II database. This set of functions enable the development of parallel FE (Finite Element) application codes and utilities for distributed memory parallel machines based on the EXODUS II database framework. This library implements a functional API which can be used in the development of advanced serial to parallel FE database utilities and MP analysis codes on the present generation of MP platforms. For this reason NEMESIS I relies on two straight forward and portable paradigms for description of the distributed finite element database. The scalar-file description relies on a single additional scalar NetCDF file that contains information on the partition of the FE database for execution on N processors. The parallel-file description utilizes a set of N files which contain the partition information for the parallel execution of the FE analysis code on N processors. This concept is illustrated in Figure 1. See Appendix A and Appendix B for specific examples.

The scalar load-balance file contains information about which nodes and elements are associated with which processor. This file also contains information about how the processors need to communicate with each other in order to obtain required boundary information. The scalar load-balance files do not generally contain geometry information, such as element connectivity, nodal coordinates or boundary condition information. This information is obtained from the original EXODUS II database. Given the scalar load-balance file, and the scalar EXODUS II database, a MP application program has all required information for execution of a parallel FE analysis code.

The scalar load-balance file, as a supplement to the scalar EXODUS II geometry file, also lends itself to a run-independent structure. Given a scalar EXODUS II database representing a problem geometry, several scalar load-balance files can be generated for runs on various numbers of processors. Since the size of a scalar load-balance file is small, relative to the EXODUS II geometry file, such files can be used to perform runs on varying number of processors depending on processor availability.

The scalar load-balance file also allows the use of a single processor to read the required information and broadcast this information to the other processors. The same processor then reads the geometry information contained in the EXODUS II file, broadcasts this information to the other processors, which then keep or discard the information depending on whether the receiving processor is responsible for that portion of the geometry. In many situations such a scheme scales better than having a single processor read, process and then send information only to the processor which requires it.

The other type of files which can be written using the NEMESIS I API are parallel geometry files. Each of these files is intended for use by a single processor. Thus, for an N-processor run there are N NEMESIS I parallel files. These files contain information necessary for a single processor to completely describe it's portion of the problem, including complete EXODUS II information and NEMESIS I information.

While it is not enforced by the NEMESIS I API, it is recommended that each of these parallel NEMESIS I files be written in such a way that they can also treated as independent EXODUS II files. This means that any utilities normally used for post processing of regular EXODUS II files can be used to view the NEMESIS I parallel files, independently. In order to conform to this recommendation, the EXODUS II information that is stored in the files must be locally numbered. Again, the user is responsible for this aspect, but the advantage of conforming to this recommendation is that each of the parallel NEMESIS I files can be treated as independent EXODUS II files. The EXODUS II API accomplishes the global-to-local mapping via the API functions `ex_put_node_num_map` and `ex_put_elem_num_map` (see [**?**]).

The NEMESIS I API is an enhanced database description, it does not contain either FE database partitioning, parallel file management routines or parallel IO capabilities. These capabilities are implemented in a number of supported and unsupported utilities that are described in [**?**]. For example, the EXODUS II database partitioning utility nem_slice uses the CHACO graph partitioning software [**?**] to produce a MP FE decomposition. This decomposition is then used to produce either a scalar-file or parallel-file description of the database. To allow generality of MP applications NEMESIS I is designed to handle both nodal and elemental based decompositions, both of which are commonly utilized by analysis codes. To accomplish this NEMESIS I makes use of generalized communication maps for both FEM node-based and element-based decompositions.

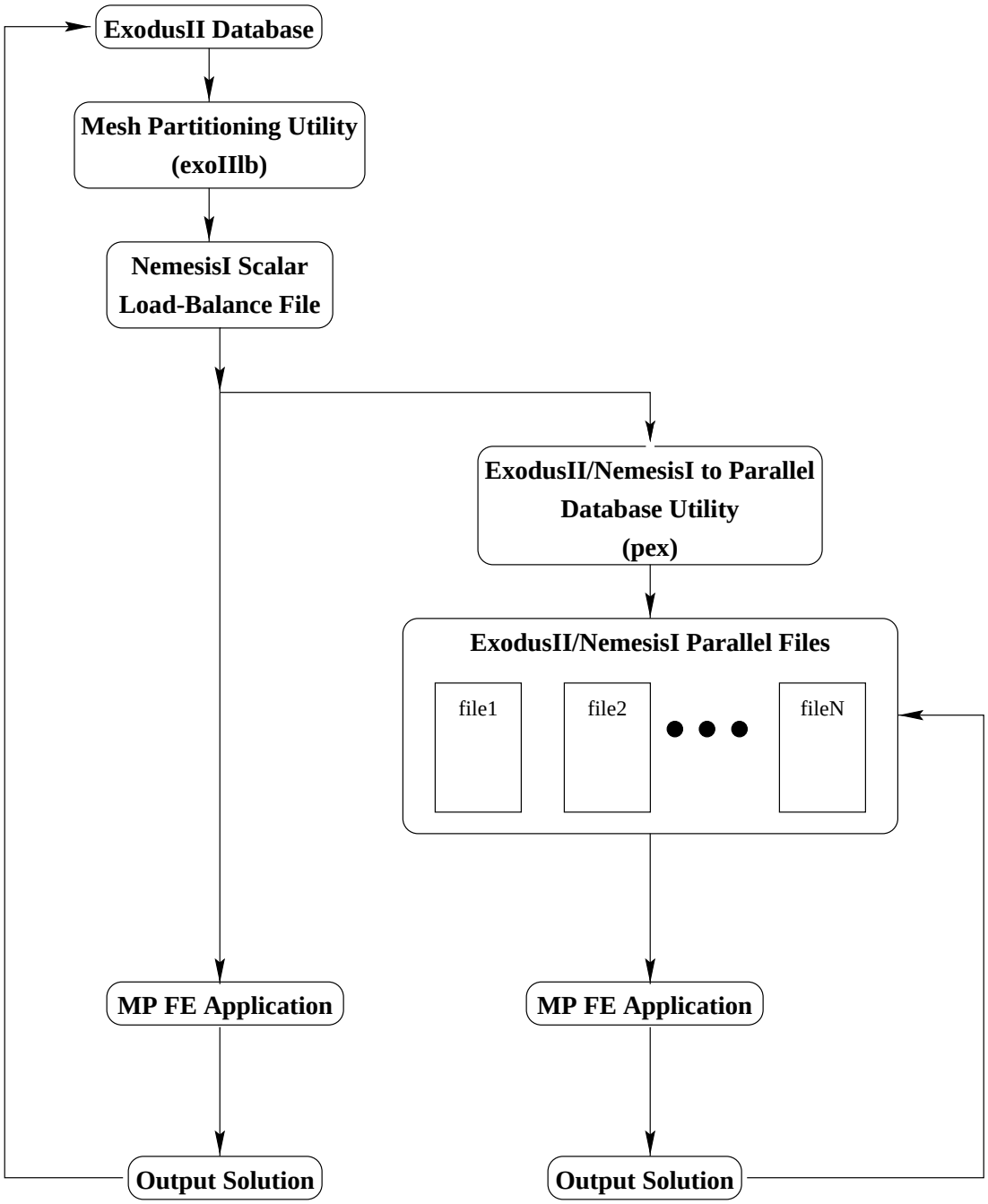

Figure 1: Conceptual description of EXODUS II /NEMESIS I parallel use.

2 The NEMESIS Application Programming Interface (API)

The NEMESIS API is similar to the API used by EXODUS II [?]. Example calling sequences for API functions are shown in Figs. 2 and 3. In these figures the *_map* functions place load balance (processor assignments) in the file and the *_cmap* functions place communication information in the file.

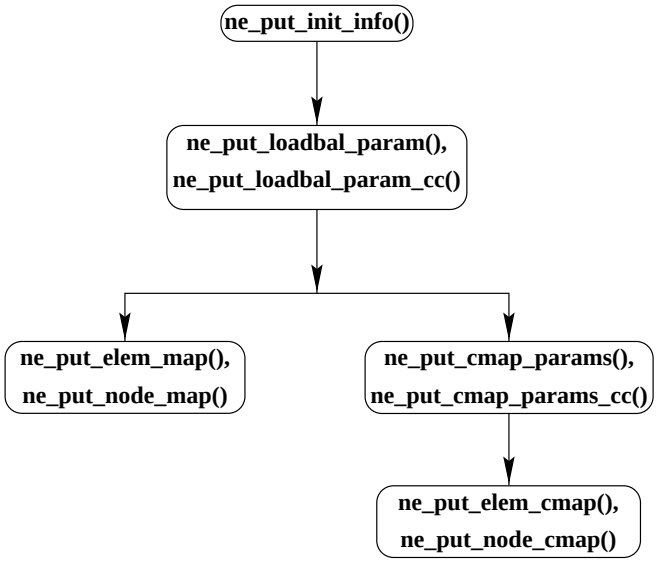

Figure 2: Calling sequence for storing load-balance information in a NEMESIS I file.

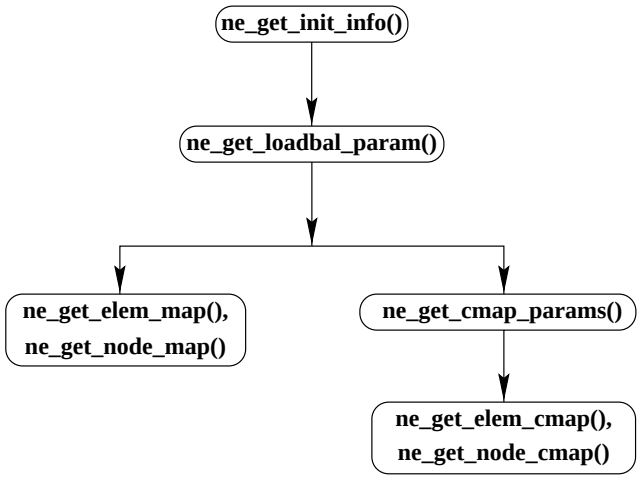

Figure 3: Calling sequence for obtaining load-balance information contained in a NEMESIS I file.

In each of the function descriptions "(R)" indicates the variable is read only while a "(W)" means that information is retrived into that variable.

2.1 Global Information

The API functions in this section read and write information about the global FEM mesh to a NEMESIS database. This information is useful to allow one of the processors to perform error checking.

2.1.1 Output Initial Global Information

The function `ne_put_init_global` outputs the global initial information. For the purposes of the function `ne_get_init_global` only *num_node_sets_global* and *num_side_sets_global* may have the value of zero.

In case of an error, `ne_put_init_global` returns a negative value.

ne_put_init_global: C Interface

```
int ne_put_init_global(neid, num_nodes_global, num_elems_global, num_elem_blks_global,
    num_node_sets_global, num_side_sets_global)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int num_nodes_global` (R)
The number of global FEM nodes.

`int num_elems_global` (R)
The number of global FEM elements.

`int num_elem_blks_global` (R)
The number of global element blocks.

`int num_node_sets_global` (R)
The number of global node sets.

`int num_side_sets_global` (R)
The number of global side sets.

2.1.2 Read Initial Global Information

The function `ne_get_init_global` reads in the global initial information.

In case of an error, `ne_get_init_global` returns a negative value.

ne_get_init_global: C Interface

```
int ne_get_init_global(neid, num_nodes_global, num_elems_global, num_elem_blks_global,
    num_node_sets_global, num_side_sets_global)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int* num_nodes_global` (W)
The number of global FEM nodes.

`int* num_elems_global` (W)
The number of global FEM elements.

`int* num_elem_blks_global` (W)
The number of global element blocks.

`int* num_node_sets_global` (W)
The number of global node sets.

`int* num_side_sets_global` (W)
The number of global side sets.

2.1.3 Output the Global Node Set Parameters

The function `ne_put_ns_param_global` outputs information about the global node sets. The function `ne_put_init_global` must be called prior to any call to `ne_put_ns_param_global`.

In case of an error, `ne_put_ns_param_global` returns a negative value.

ne_put_ns_param_global: C Interface

```
int ne_put_ns_param_global(neid, global_ids, num_global_node_counts,
    num_global_df_counts)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int* global_ids` (R)
Pointer to a vector of global node set IDs.

`int* num_global_node_counts` (R)
Pointer to a vector of global FEM node counts contained in each global node set.

`int* num_global_df_counts` (R)
Pointer to a vector of global distribution factor counts in each global node set.

2.1.4 Read the Global Node Set Parameters

The function `ne_get_ns_param_global` reads information about the global node sets. Memory for the vectors *global_ids*, *num_global_node_counts* and *num_global_df_counts* must be allocated prior to any call to `ne_get_ns_param_global`.

In case of an error, `ne_get_ns_param_global` returns a negative value.

ne_get_ns_param_global: C Interface

```
int ne_get_ns_param_global(neid, global_ids, num_global_node_counts,
    num_global_df_counts)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int* global_ids` (W)
Pointer to a vector of global node set IDs.

`int* num_global_node_counts` (W)
Pointer to a vector of global FEM node counts in each global node set.

`int* num_global_df_counts` (W)
Pointer to a vector of global distribution factor counts in each global node set.

2.1.5 Output the Global Side Set Parameters

The function `ne_put_ss_param_global` outputs information about the global side sets. The function `ne_put_init_global` must be called prior to any call to `ne_put_ss_param_global`.

In case of an error, `ne_put_ss_param_global` returns a negative value.

ne_put_ss_param_global: C Interface

```
int ne_put_ss_param_global(neid, global_ids, num_global_side_counts,
    num_global_df_counts)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int* global_ids` (R)
Pointer to a vector of global side set IDs.

`int* num_global_side_counts` (R)
Pointer to a vector of global FEM side/element counts in each global side set.

`int* num_global_df_counts` (R)
Pointer to a vector of global distribution factor counts in each global side set.

2.1.6 Read the Global Side Set Parameters

The function `ne_get_ss_param_global` reads information about the global side sets. Memory for *global_ids*, *num_global_side_counts* and *num_global_df_counts* must be allocated prior to calling `ne_get_ss_param_global`.

In case of an error, `ne_get_ss_param_global` returns a negative value.

ne_get_ss_param_global: C Interface

```
int ne_get_ss_param_global(neid, global_ids, num_global_side_counts,
    num_global_df_counts)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int* global_ids` (W)
Pointer to a vector of global side set IDs.

`int* num_global_side_counts` (W)
Pointer to a vector of global FEM side/element counts in each global side set.

`int* num_global_df_counts` (W)
Pointer to a vector of global distribution factor counts in each global side set.

2.1.7 Output the Global Element Block Information

The function `ne_put_eb_info_global` outputs information about the global element blocks. The function `ne_put_init_global` must be called prior to any call `ne_put_eb_info_global`.

In case of an error, `ne_put_eb_info_global` returns a negative value.

ne_put_eb_info_global: C Interface

```
int ne_put_eb_info_global(neid, global_elem_blk_ids, global_elem_blk_cnts)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int* global_elem_blk_ids` (R)
Pointer to a vector of global element block IDs.

`int* global_elem_blk_cnts` (R)
Pointer to a vector of global element block counts.

2.1.8 Read the Global Element Block Information

The function `ne_get_eb_info_global` reads information about the global element blocks. Memory for *global_elem_blk_ids* and *global_elem_blk_cnts* must be allocated prior to calling `ne_get_eb_info_global`.

In case of an error, `ne_get_eb_info_global` returns a negative value.

ne_get_eb_info_global: C Interface

```
int ne_get_eb_info_global(neid, global_elem_blk_ids, global_elem_blk_cnts)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int* global_elem_blk_ids` (W)
Pointer to a vector of global element block IDs.

`int* global_elem_blk_cnts` (W)
Pointer to a vector of global element block counts.

2.2 Decomposition Information

The API functions in this section read and write information about how the original geometry was decomposed for the parallel run.

2.2.1 Output Initial Information

The function `ne_put_init_info` outputs some initial information.

In case of an error, `ne_put_init_info` returns a negative value.

ne_put_init_info: C Interface

```
int ne_put_init_info(neid, num_proc, num_proc_in_file, file_type)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int num_proc` (R)
The number of processors for which the NEMESIS I file was created.

`int num_proc_in_file` (R)
The number of processors for which the NEMESIS I file stores information. This is generally equal to `num_proc`.

`char* ftype` (R)
The type of file to be written. Either *"s"*, for a scalar load-balance file, or *"p"* for a parallel file.

2.2.2 Read Initial Information

The function `ne_get_init_info` reads some initial information.

In case of an error, `ne_get_init_info` returns a negative value.

ne_get_init_info: C Interface

```
int ne_get_init_info(neid, num_proc, num_proc_in_file, file_type)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int* num_proc` (W)
The number of processors for which the NEMESIS I file was created.

`int* num_proc_in_file` (W)
The number of processors for which the NEMESIS I file stores information. This is generally equal to `num_proc`.

`char* ftype` (W)
The type of file to be written. Either *"s"*, for a scalar load-balance file, or *"p"* for a parallel file.

2.2.3 Output the Load Balance Parameters

The function `ne_put_loadbal_param` outputs parameters necessary to describe the decomposition. This function, or `ne_put_loadbal_param_cc`, must be called prior to any other API functions which output decomposition information.

NOTE: This function is no longer valid for use with scaler load balance files. The function, `ne_put_loadbal_param_cc`, should used.

In case of an error, `ne_put_loadbal_param` returns a negative value.

ne_put_loadbal_param: C Interface

```
int ne_put_loadbal_param(neid, num_internal_nodes, num_border_nodes, num_external_nodes,
    num_internal_elems, num_border_elems, num_node_cmaps, num_elem_cmaps, processor)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int num_internal_nodes` (R)
The number of FEM nodes contained in FEM elements wholly owned by the current processor.

`int num_border_nodes` (R)
The number of FEM nodes local to a processor but residing in an element which also has FEM nodes on other processors.

`int num_external_nodes` (R)
The number of FEM nodes that reside on other processors but whose element partially resides on the current processor.

`int num_internal_elems` (R)
The number of internal FEM elements. Elements local to this processor.

`int num_border_elems` (R)
The number of border FEM elements. Elements local to this processor but whose FEM nodes reside on other processors as well.

`int num_node_cmaps` (R)
The number of nodal communication maps for this processor.

`int num_elem_cmaps` (R)
The number of elemental communication maps for this processor.

`int processor` (R)
The processor ID of the processor for which information is to be written.

2.2.4 Output Concatenated Load Balance Parameters

The function `ne_put_loadbal_param_cc` outputs the load-balance parameters for all processors simultaneously. Each input array is the size of *num_proc_in_file*. This function, or `ne_put_loadbal_param`, must be called prior to a call to other decomposition functions.

In case of an error, `ne_put_loadbal_param_cc` returns a negative number.

ne_put_loadbal_param_cc: C Interface

```
int ne_put_loadbal_param_cc(neid, num_internal_nodes, num_border_nodes,
    num_external_nodes, num_internal_elems, num_border_elems, num_node_cmaps,
    num_elem_cmaps)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int* num_internal_nodes` (R)
Pointer to a vector containing the number of internal FEM nodes for each *num_proc_in_file* processor.

`int* num_border_nodes` (R)
Pointer to a vector containing the number of border FEM nodes for each *num_proc_in_file* processor.

`int* num_external_nodes` (R)
Pointer to a vector containing the number of external FEM nodes for each *num_proc_in_file* processor.

`int* num_internal_elems` (R)
Pointer to a vector containing the number of internal FEM elements for each *num_proc_in_file* processor.

`int* num_border_elems` (R)
Pointer to a vector containing the number of border FEM elements for each *num_proc_in_file* processor.

`int* num_node_cmap` (R)
Pointer to a vector containing the number of nodal communication maps for each *num_proc_in_file* processor.

`int* num_elem_cmap` (R)
Pointer to a vector containing the number elemental communication maps for each *num_proc_in_file* processor.

2.2.5 Read the Load Balance Parameters

The function `ne_get_loadbal_param` reads parameters necessary to describe the decomposition. Memory for *ftype* must be allocated prior to any call to `ne_get_loadbal_param`.

In case of an error, `ne_get_loadbal_param` returns a negative value.

ne_get_loadbal_param: C Interface

```
int ne_get_loadbal_param(neid, num_internal_nodes, num_border_nodes, num_external_nodes,
    num_internal_elems, num_border_elems, num_node_cmaps, num_elem_cmaps, processor)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int* num_internal_nodes` (W)
The number of FEM nodes contained in FEM elements wholly owned by the current processor.

`int* num_border_nodes` (W)
The number of FEM nodes local to a processor but residing in an element which also has FEM nodes on other processors.

`int* num_external_nodes` (W)
The number of FEM nodes that reside on other processor but whose element partially resides on the current processor.

`int* num_internal_elems` (W)
The number of internal FEM elements. Elements local to this processor.

`int* num_border_elems` (W)
The number of border FEM elements. Elements local to this processor but whose FEM nodes reside on other processors as well.

`int* num_node_cmaps` (W)
The number of nodal communication maps for this processor.

`int* num_elem_cmaps` (W)
The number of elemental communication maps for this processor.

`int processor` (R)
The processor ID of the processor for which information is to be read.

2.2.6 Output the Element Map

The function `ne_put_elem_map` outputs information about how the elements contained in the database are to be interpreted. Elements are either "internal", local to a processor, or "border", shared by processors. This information is generally used by elemental-based decomposition codes. The function `ne_put_loadbal_param`, or `ne_put_loadbal_param_cc`, must be called prior to any call to `e_put_elem_map`.

In case of an error, `ne_put_elem_map` returns a negative number.

ne_put_elem_map: C Interface

```
int ne_put_elem_map(neid, elem_mapi, elem_mapb, processor)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int* elem_mapi` (R)
Pointer to a vector containing the element IDs for the internal elements residing on this processor.

`int* elem_mapb` (R)
Pointer to a vector containing the element IDs for the border elements residing on this processor.

`int processor` (R)
The processor ID of the processor for which information is to be output.

2.2.7 Read the Element Map

The function `ne_get_elem_map` reads information about how the elements contained in the database are to be interpreted. Memory must be allocated for *elem_mapi* and *elem_mapb* prior to any call to `ne_get_elem_map`.

In case of an error, `ne_get_elem_map` returns a negative value.

ne_get_elem_map: C Interface

```
int ne_get_elem_map(neid, elem_mapi, elem_mapb, processor)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int* elem_mapi` (W)
Pointer to a vector into which the element IDs for the internal elements residing on this processor will be read.

`int* elem_mapb` (W)
Pointer to a vector into which the element IDs for the border elements residing on this processor will be read.

`int processor` (R)
The processor ID of the processor for which information is to be read.

2.2.8 Output the Node Map

The function `ne_put_node_map` outputs information about how the nodes contained in the database are to be interpreted. Nodes are either "internal", local to a processor, "border", local to a processor, but external to other processors, or "external", needed by the current processor but owned by another processor. A call to `ne_put_loadbal_param`, or `ne_put_loadbal_param_cc`, must be made prior to any call to `ne_put_node_map`.

In case of an error, `ne_put_node_map` returns a negative value.

ne_put_node_map: C Interface

```
int ne_put_node_map(neid, node_mapi, node_mapb, node_mape, processor)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int* node_mapi` (R)
Pointer to a vector containing the local node IDs of "internal" FEM nodes.

`int* node_mapb` (R)
Pointer to a vector containing the local node IDs of "border" FEM nodes.

`int* node_mape` (R)
Pointer to a vector containing the local node IDs of "external" FEM nodes.

`int processor` (R)
The processor ID of the processor for which information is to be written.

2.2.9 Read the Node Map

The function `ne_get_node_map` reads information about how the nodes contained in the database are to be interpreted. Memory must be allocated for *node_mapi*, `node_mapb` and `node_mape` prior to the call to `ne_get_node_map`.

In case of an error, `ne_get_node_map` returns a negative value.

ne_get_node_map: C Interface

```
int ne_get_node_map(neid, node_mapi, node_mapb, node_mape, processor)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int* node_mapi` (W)
Pointer to a vector where the local node IDs of "internal" FEM nodes are to be stored.

`int* node_mapb` (W)
Pointer to a vector where the local node IDs of "border" FEM nodes are to be stored.

`int* node_mape` (W)
Pointer to a vector where the local node IDs of "external" FEM nodes are to be stored.

`int processor` (R)
The processor ID of the processor for which information is to be read.

2.3 Communication Information

The API functions in this section handle both nodal (for node based decompositions) and elemental (for elemental based decompositions) communication maps.

2.3.1 Output the Communication Map Parameters

The function `ne_put_cmap_params` outputs the initial information required for communication maps. This function, or `ne_put_cmap_params_cc`, must be called prior to calling any other API functions which write communication map information. The function `ne_put_loadbal_params`, or `ne_put_loadbal_params_cc` must be called prior to any call to `ne_put_cmap_params`.

NOTE: This function is no longer valid for use with scaler load balance files. The function, `ne_put_loadbal_param_cc`, should used.

In case of an error, `ne_put_cmap_params` returns a negative value.

ne_put_cmap_params: C Interface

```
int ne_put_cmap_params(neid, node_cmap_ids, node_cmap_node_cnts, elem_cmap_ids,
    elem_cmap_elem_cnts, processor)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int* node_cmap_ids` (R)
Pointer to a vector containing the map IDs for each nodal communication map.

`int* node_cmap_node_cnts` (R)
Pointer to a vector containing the node counts for each nodal communication map.

`int* elem_cmap_ids` (R)
Pointer to a vector containing the map IDs for each elemental communication map.

`int* elem_cmap_elem_cnts` (R)
Pointer to a vector containing the element counts for each elemental communication map.

`int processor` (R)
The processor ID of the processor for which information is to be written.

2.3.2 Output Concatenated Communication Map Parameters

The function ne_put_cmap_params_cc outputs communication map information for all processors in a single call. This function, or ne_put_cmap_params, must be called prior to calling any other API function which output communication map information. The function ne_put_loadbal_params, or ne_put_loadbal_params_cc, must be called prior to any call to ne_put_cmap_params_cc.

In case of an error, ne_put_cmap_params_cc returns a negative value.

ne_put_cmap_params_cc: C Interface

```
int ne_put_cmap_params_cc(neid, node_cmap_ids, ne_cmap_node_cnts, node_proc_ptrs,
    elem_cmap_ids, elem_cmap_elem_cnts, elem_proc_ptrs)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to ex_create or ex_open.

`int* node_cmap_ids` (R)
Pointer to a vector of nodal communication map IDs for all the processors.

`int* node_cmap_node_cnts` (R)
Pointer to a vector of FEM node counts in each nodal communication map for all the processors.

`int* node_proc_ptrs` (R)
Pointer to a vector which is used to index into the nodal communication map vectors.

`int* elem_cmap_ids` (R)
Pointer to a vector of elemental communication map IDs for all the processors.

`int* elem_cmap_elem_cnts` (R)
Pointer to a vector of FEM elem counts in each elemental communication map for all the processors.

`int* elem_proc_ptrs` (R)
Pointer to a vector which is used to index into the elemental communication map vectors.

2.3.3 Read the Communication Map Parameters

The function `ne_get_cmap_params` reads in parameters for each of the communication maps. Memory for each of the parameter vectors must be allocated prior to a call to `ne_get_cmap_params`. The number of nodal and elemental maps contained in the Nemesis file, for a given processor, can be obtained via a call to `ne_get_loadbal_param`.

In case of an error, `ne_get_cmap_params` returns a negative value.

ne_get_cmap_params: C Interface

```
int ne_get_cmap_params(neid, node_cmap_ids, node_cmap_node_cnts, elem_cmap_ids,
    elem_cmap_elem_cnts, processor)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int* node_cmap_ids` (W)
Pointer to a vector into which the ID of each nodal communication map will be stored.

`int* node_cmap_node_cnts` (W)
Pointer to a vector into which FEM node counts for each nodal communication map will be stored.

`int* elem_cmap_ids` (W)
Pointer to a vector into which the ID of each elemental communication map will be stored.

`int* elem_map_elem_cnts` (W)
Pointer to a vector into which FEM element counts for each elemental communication map will be stored.

`int processor` (R)
The processor ID of the processor for which information is to be written.

2.3.4 Output a Nodal Communication Map

The function `ne_put_node_cmap` outputs a nodal communication map. The function `ne_put_cmap_params`, or `ne_put_cmap_params_cc`, must be called prior to any call to `ne_put_node_cmap`.

In case of an error, `ne_put_node_cmap` returns a negative value.

ne_put_node_cmap: C Interface

```
int ne_put_node_cmap(neid, map_id, node_ids, proc_ids, processor)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int map_id` (R)
The ID of the nodal communication map to be output.

`int* node_ids` (R)
The nodal IDs of the FEM nodes in this communication map.

`int* proc_ids` (R)
The processor IDs associated with each of the FEM nodes in *node_ids*.

`int processor` (R)
The processor ID of the processor for which information is to be read.

2.3.5 Read a Nodal Communication Map

The function `ne_get_node_cmap` reads a nodal communication map with the specified ID. Memory for *node_ids* and *proc_ids* must be allocated prior to calling this function.

In case of an error, `ne_get_node_cmap` returns a negative value.

ne_get_node_cmap: C Interface

```
int ne_get_node_cmap(neid, map_id, node_ids, proc_ids, processor)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int map_id` (R)
The ID of the nodal communication map to read.

`int* node_ids` (W)
The nodal IDs of the FEM nodes in this communication map.

`int* proc_ids` (W)
The processor IDs associated with each of the FEM nodes in *node_ids*

`int processor` (R)
The processor ID of the processor for which information is to be read.

2.3.6 Output an Elemental Communication Map

The function `ne_put_elem_cmap` outputs information about the specified elemental communication map. The function `ne_put_cmap_params`, or `ne_put_cmap_params_cc`, must be called prior to any call to `ne_put_elem_cmap`.

In case of an error, `ne_put_elem_cmap` returns a negative value.

ne_put_elem_cmap: C Interface

```
int ne_put_elem_cmap(neid, map_id, elem_ids, side_ids, proc_ids, processor)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int map_id` (R)
The ID of the elemental communication map to be output.

`int* elem_ids` (R)
Pointer to a vector containing the element IDs of the elements contained in this elemental communication map.

`int* side_ids` (R)
Pointer to a vector containing the side IDs for each of the elements in *elem_ids*.

`int* proc_ids` (R)
Pointer to a vector containing the processor IDs of each of the elements in *elem_ids*.

`int processor` (R)
The processor ID of the processor for which information is to be output.

2.3.7 Read an Elemental Communication Map

The function `ne_get_elem_cmap` reads the information for the specified elemental communication map. Memory for the vectors *elem_ids* and *side_ids* must be allocated prior to calling `ne_get_elem_cmap`.

In case of an error, `ne_get_elem_cmap` returns a negative value.

ne_get_elem_cmap: C Interface

```
int ne_get_elem_cmap(neid, map_id, elem_ids, side_ids, proc_ids, processor)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int map_id` (R)
Elemental communication map ID of map to retrieve.

`int* elem_ids` (W)
Pointer to a vector in which the element IDs of the elements in this communication map will be returned.

`int* side_ids` (W)
Pointer to a vector in which the side IDs, for each element listed in *elem_ids*, will be returned.

`int* proc_ids` (W)
Pointer to a vector in which the processor IDs, for each element listed in *elem_ids*, will be returned.

`int processor` (R)
The processor ID of the processor for which information is to be read.

2.4 Partial Read Function

The following API functions are used to perform reads of partial information from an EXODUS II database. The functions purpose is enable a single processor to read information from the EXODUS II database when it otherwise might not be possible due to memory constraints.

2.4.1 Read *n* Coordinate Values

The function `ne_get_n_coord` reads the specified number of coordinate values starting at the specified index. Memory for the vectors *x_coor*, *y_coor* and *z_coor* must be allocated prior to any call to `ne_get_n_coord`.

In case of an error, `ne_get_n_coord` returns a negative value.

ne_get_n_coord: C Interface

```
int ne_get_n_coord(neid, start_node_num, num_nodes, x_coor, y_coor, z_coor)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int start_node_num` (R)
The position in the coordinate list from which to start reading nodal coordinates from the EXODUS II file.

`int num_nodes` (R)
The number of nodal coordinates to read, starting at *start_node_num*.

`float* x_coor` (W)
Vector for storage of the X nodal coordinates.

`float* y_coor` (W)
Vector for storage of the Y nodal coordinates.

`float* z_coor` (W)
Vector for storage of the Z nodal coordinates.

2.4.2 Read Information for n Entries in a Nodal Variable Vector

The function `ne_get_n_nodal_var` reads information for the specified number of FEM nodes from the specified nodal variable, starting at the specified location.

In case of an error, `ne_get_n_nodal_var` returns a negative value.

ne_get_n_nodal_var: C Interface

```
int ne_get_n_nodal_var(neid, time_step, nodal_var_index, start_node_num, num_nodes,
    nodal_var_vals)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int time_step` (R)
The index of the time step from which to retrieve nodal variable(s).

`int nodal_var_index` (R)
The nodal variable from which information is to be obtained.

`int start_node_num` (R)
Location from which to start reading the *nodal_var_vals* in the time index *start_node_num* .

`int num_nodes` (R)
The number of entries in the nodal variable to read.

`void* nodal_var_vals` (W)
Pointer to a vector into which the values of the nodal variables will be placed.

2.4.3 Read Information for n Entries in the Node Number Map

The function `ne_get_n_node_num_map` reads information for the specified number of FEM nodes from the node number map.

In case of an error, `ne_get_n_node_num_map` returns a negative value.

ne_get_n_node_num_map: C Interface

```
int ne_get_n_node_num_map(neid, start_node_num, num_nodes, node_map)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int start_node_num`
Location from which to start reading the *nodal_map* .

`int num_nodes` (R)
The number of entries in the nodal number map to read.

`int* node_map`
Pointer to a vector into which the partial node number map will be read.

2.4.4 Read Connectivity Information for *n* Elements

The function `ne_get_n_elem_conn` reads connectivity information for the specified number of elements, starting at the specified location, and in the specified element block. Memory for *connect* must be allocated prior to calling `ne_get_n_elem_conn`.

In case of an error, `ne_get_n_elem_conn` returns a negative value.

ne_get_n_elem_conn: C Interface

```
int ne_get_n_elem_conn(neid, elem_blk_id, start_elem_num, num_elems, connect)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int elem_blk_id` (R)
The element block ID from which the connectivity is to be read.

`int start_elem_num` (R)
The number of the element where the read of the connectivity should start, within the element block whose ID is *elem_blk_id*.

`int num_elems` (R)
The number of elements to read the connectivity for, start at element number *start_elem_num*.

`int* connect` (W)
Pointer to a vector into which the connectivity is to be read.

2.4.5 Read *n* Element Attributes

The function `ne_get_n_elem_attr` reads the elemental attributes for the specified number of elements starting at the specified element ID in the specified element block. Memory for *attrib* must be allocated prior to calling `ne_get_n_elem_attr`.

In case of an error, `ne_get_n_elem_attr` returns a negative value.

ne_get_n_elem_attr: C Interface

```
int ne_get_n_elem_attr(neid, elem_blk_id, start_elem_num, num_elems, attrib)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int elem_blk_id` (R)
The element block ID from which the attributes are to be read.

`int start_elem_num` (R)
The element number where reading of the attributes will begin, within the element block whose ID is *elem_blk_id*.

`int num_elems` (R)
The number of elements to read the attributes for, starting at element number *start_elem_num*.

`void* attrib` (W)
Pointer to a vector into which the attributes are to be read.

2.4.6 Read Information for n Entries in a Elemental Variable Vector

The function `ne_get_n_elem_var` reads information for the specified number of elements from the specified elemental variable in the specified element block, and starting at the specified location.

In case of an error, `ne_get_n_elem_var` returns a negative value.

ne_get_n_elem_var: C Interface

```
int ne_get_n_elem_var(neid, time_step, elem_var_index, elem_blk_id, num_elem_this_blk,
    start_elem_num, num_elems, elem_var_vals)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int time_step` (R)
The index of the time step from which to retrieve elemental variable(s).

`int elem_var_index` (R)
The elemental variable from which information is to be obtained.

`int elem_blk_id` (R)
The elemental block id from which information is to be obtained.

`int num_elem_this_blk` (R)
The number of elementals in this block.

`int start_elem_num` (R)
Location from which to start reading the *elem_var_vals* in the time index *start_elem_num* .

`int num_elem` (R)
The number of entries in the elemental variable to read.

`void* elem_var_vals` (W)
Pointer to a vector into which the values of the elemental variables will be placed.

2.4.7 Read Information for n Entries in the Elemental Number Map

The function `ne_get_n_elem_num_map` reads information for the specified number of elements from the elemental number map.

In case of an error, `ne_get_n_elem_num_map` returns a negative value.

ne_get_n_elem_num_map: C Interface

```
int ne_get_n_elem_num_map(neid, start_elem_num, num_ents, elem_map)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int start_elem_num`
Location from which to start reading the *elem_map* .

`int num_ents` (R)
The number of entries in the elemental number map to read.

`int* elem_map`
Pointer to a vector into which the partial elemental number map will be read.

2.4.8 Read Information for n Sides in a Side Set

The function `ne_get_n_side_set` reads information for the specified number of sides in the specified side set, starting at the specified location.

In case of an error, `ne_get_n_side_set` returns a negative value.

ne_get_n_side_set: C Interface

```
int ne_get_n_side_set(neid, side_set_id, start_side_num, num_sides, side_set_elem_list,
    side_set_side_list)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int side_set_id` (R)
ID of the side set from which info is to be gathered.

`int start_side_num` (R)
Location from which to start reading the *side_set_elem_list* and *side_set_side_list* for side set ID *side_set_id*.

`int num_sides` (R)
The number of sides to read into *side_set_elem_list* and *side_set_side_list*.

`int* side_set_elem_list` (W)
Vector into which the element list for the specified side set is to be read.

`int* side_set_side_list` (W)
Vector into which the side list for the specified side set is to be read.

2.4.9 Read Distribution Factors for *n* Sides in a Side Set

The function `ne_get_n_side_set_df` reads distribution factors for the specified number of sides in the specified side set, starting at the specified location.

In case of an error, `ne_get_n_side_set_df` returns a negative value.

ne_get_n_side_set_df: C Interface

```
int ne_get_n_side_set_df(neid, side_set_id, start_side_num, num_df_to_get, side_set_df, )
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int side_set_id` (R)
ID of the side set from which distribution factors are to be gathered.

`int start_side_num` (R)
Location from which to start reading the *side_set_df* for side set ID *side_set_id*.

`int num_df_to_get` (R)
The number of distribution factors to read into *side_set_df*.

`void* side_set_df` (W)
Vector into which the distribution factors for the specified side set are to be read.

2.4.10 Read Information for *n* Nodes in a Node Set

The function `ne_get_n_node_set` reads information for the specified number of nodes in the specified node set, starting at the specified location.

In case of an error, `ne_get_n_node_set` returns a negative value.

ne_get_n_node_set: C Interface

```
int ne_get_n_node_set(neid, node_set_id, start_node_num, num_nodes, node_set_node_list)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int node_set_id` (R)
ID of the node set from which info is to be gathered.

`int start_node_num` (R)
Location from which to start reading the *node_set_node_list* for node set ID *node_set_id*.

`int num_nodes` (R)
The number of nodes to read into *node_set_node_list*.

`int* node_set_node_list` (W)
Vector into which the node list for the specified node set is to be read.

2.4.11 Read Distribution Factors for *n* Nodes in a Node Set

The function `ne_get_n_node_set_df` reads distribution factors for the specified number of nodes in the specified node set, starting at the specified location.

In case of an error, `ne_get_n_node_set_df` returns a negative value.

ne_get_n_node_set_df: C Interface

```
int ne_get_n_node_set_df(neid, node_set_id, start_node_num, num_df_to_get, node_set_df, )
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int node_set_id` (R)
ID of the node set from which distribution factors are to be gathered.

`int start_node_num` (R)
Location from which to start reading the *node_set_df* for node set ID *node_set_id*.

`int num_df_to_get` (R)
The number of distribution factors to read into *node_set_df*.

`void* node_set_df` (W)
Vector into which the distribution factors for the specified node set are to be read.

2.5 Partial Write Function

The following API functions are used to perform writes of partial information to an EXODUS II database. The functions purpose is to enable a single processor to write information from the NEMESIS I database files to a serial EXODUS II database when it otherwise might not be possible due to memory constraints.

2.5.1 Output *n* Coordinate Values

The function `ne_put_n_coord` writes the specified number of coordinate values starting at the specified index.

In case of an error, `ne_put_n_coord` returns a negative value.

ne_put_n_coord: C Interface

```
int ne_put_n_coord(neid, start_node_num, num_nodes, x_coor, y_coor, z_coor)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int start_node_num` (R)
The position in the coordinate list from which to start writing nodal coordinates to the EXODUS II file.

`int num_nodes` (R)
The number of nodal coordinates to write, starting at *start_node_num*.

`float* x_coor` (W)
Vector containing the X nodal coordinates.

`float* y_coor` (W)
Vector containing the Y nodal coordinates.

`float* z_coor` (W)
Vector containing the Z nodal coordinates.

2.5.2 Output a Nodal Variable Slab

The function `ne_put_nodal_var_slab` outputs a slab of nodal variables with the given index, starting at the specified location.

In case of an error, `ne_put_nodal_var_slab` returns a negative value.

ne_put_nodal_var_slab: C Interface

```
int ne_put_nodal_var_slab(neid, time_step, nodal_var_index, start_pos, num_vals,
    nodal_var_vals)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`time_step` (R)
The time step index where nodal variables are to be written.

`nodal_var_index` (R)
The nodal variable into which data is to be written.

`start_pos` (R)
The location, in the nodal variable vector within the EXODUS II file, where the data is to be written.

`num_vals` (R)
The number of values to output.

`void* nodal_var_vals` (R)
Pointer to a vector containing the values of the nodal variables.

2.5.3 Output Information for n Entries in the Node Number Map

The function `ne_put_n_node_num_map` writes information for the specified number of FEM nodes to the node number map.

In case of an error, `ne_put_n_node_num_map` returns a negative value.

ne_put_n_node_num_map: C Interface

```
int ne_put_n_node_num_map(neid, start_node_num, num_nodes, node_map)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int start_node_num`
Location at which to start writing the *nodal_map* .

`int num_nodes` (R)
The number of entries in the nodal number map to write.

`int* node_map`
Pointer to a vector containing the partial node number map that will be written.

2.5.4 Output Connectivity Information for *n* Elements

The function `ne_put_n_elem_conn` writes connectivity information for the specified number of elements, starting at the specified location, and in the specified element block.

In case of an error, `ne_put_n_elem_conn` returns a negative value.

ne_put_n_elem_conn: C Interface

```
int ne_put_n_elem_conn(neid, elem_blk_id, start_elem_num, num_elems, connect)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int elem_blk_id` (R)
The element block ID for which the connectivity is to be written.

`int start_elem_num` (R)
The number of the element where the write of the connectivity should start, within the element block whose ID is *elem_blk_id*.

`int num_elems` (R)
The number of elements to write the connectivity for, starting at element number *start_elem_num*.

`int* connect` (W)
Pointer to a vector containing the connectivity to be written.

2.5.5 Output *n* Element Attributes

The function `ne_put_n_elem_attr` writes the elemental attributes for the specified number of elements starting at the specified element ID in the specified element block.

In case of an error, `ne_put_n_elem_attr` returns a negative value.

ne_put_n_elem_attr: C Interface

```
int ne_put_n_elem_attr(neid, elem_blk_id, start_elem_num, num_elems, attrib)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int elem_blk_id` (R)
The element block ID for which the attributes are to be written.

`int start_elem_num` (R)
The element number where writing of the attributes will begin, within the element block whose ID is *elem_blk_id*.

`int num_elems` (R)
The number of elements to write the attributes for, starting at element number *start_elem_num*.

`void* attrib` (W)
Pointer to a vector containing the attributes are to be written.

2.5.6 Output a Elemental Variable Slab

The function `ne_put_elem_var_slab` outputs a slab of elemental variables with the given index, starting at the specified location in the specified element block.

In case of an error, `ne_put_elem_var_slab` returns a negative value.

ne_put_elem_var_slab: C Interface

```
int ne_put_elem_var_slab(neid, time_step, elem_var_index, elem_blk_id, start_pos,
    num_vals, elem_var_vals)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`time_step` (R)
The time step index where elemental variables are to be written.

`elem_var_index` (R)
The elemental variable into which data is to be written.

`int elem_blk_id` (R)
The element block ID for which the variable is to be written.

`start_pos` (R)
The location, in *elem_blk_id* of the elemental variable vector within the EXODUS II file, where the data is to be written.

`num_vals` (R)
The number of values to output.

`void* elem_var_vals` (R)
Pointer to a vector containing the values of the elemental variables.

2.5.7 Output Information for n Entries of the Elemental Number Map

The function `ne_put_n_elem_num_map` writes information for the specified number of elements to the elemental number map.

In case of an error, `ne_put_n_elem_num_map` returns a negative value.

ne_put_n_elem_num_map: C Interface

```
int ne_put_n_elem_num_map(neid, start_elem_num, num_ents, elem_map)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int start_elem_num`
Location at which to start writing the *elem_map* .

`int num_ents` (R)
The number of entries in the elemental number map to write.

`int* elem_map`
Pointer to a vector containing the partial elemental number map that will be written.

2.5.8 Output Information for n Sides in a Side Set

The function `ne_put_n_side_set` writes information for the specified number of sides in the specified side set, starting at the specified location.

In case of an error, `ne_put_n_side_set` returns a negative value.

ne_put_n_side_set: C Interface

```
int ne_put_n_side_set(neid, side_set_id, start_side_num, num_sides, side_set_elem_list,
    side_set_side_list)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int side_set_id` (R)
ID of the side set to which info is to be written.

`int start_side_num` (R)
Location at which to start writing the *side_set_elem_list* and *side_set_side_list* for side set ID *side_set_id*.

`int num_sides` (R)
The number of sides to write from *side_set_elem_list* and *side_set_side_list*.

`int* side_set_elem_list` (W)
Vector containing the element list for the specified side set to be written.

`int* side_set_side_list` (W)
Vector containing the side list for the specified side set to be written.

2.5.9 Output Distribution Factors for *n* Sides in a Side Set

The function `ne_put_n_side_set_df` writes distribution factors for the specified number of sides in the specified side set, starting at the specified location.

In case of an error, `ne_put_n_side_set_df` returns a negative value.

ne_put_n_side_set_df: C Interface

```
int ne_put_n_side_set_df(neid, side_set_id, start_side_num, num_df_to_get, side_set_df, )
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int side_set_id` (R)
ID of the side set to which distribution factors are to be written.

`int start_side_num` (R)
Location at which to start writing the *side_set_df* for side set ID *side_set_id*.

`int num_df_to_get` (R)
The number of distribution factors to written from *side_set_df*.

`void* side_set_df` (W)
Vector containing the distribution factors for the specified side set to be written.

2.5.10 Output Information for n Nodes in a Node Set

The function `ne_put_n_node_set` writes information for the specified number of nodes in the specified node set, starting at the specified location.

In case of an error, `ne_put_n_node_set` returns a negative value.

ne_put_n_node_set: C Interface

```
int ne_put_n_node_set(neid, node_set_id, start_node_num, num_nodes, node_set_node_list)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int node_set_id` (R)
ID of the node set to which info is to be written.

`int start_node_num` (R)
Location at which to start writing the *node_set_node_list* for node set ID *node_set_id*.

`int num_nodes` (R)
The number of nodes to written from *node_set_node_list*.

`int* node_set_node_list` (W)
Vector containing the node list for the specified node set is to be written.

2.5.11 Output Distribution Factors for *n* Nodes in a Node Set

The function `ne_put_n_node_set_df` writes distribution factors for the specified number of nodes in the specified node set, starting at the specified location.

In case of an error, `ne_put_n_node_set_df` returns a negative value.

ne_put_n_node_set_df: C Interface

```
int ne_put_n_node_set_df(neid, node_set_id, start_node_num, num_df_to_get, node_set_df, )
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int node_set_id` (R)
ID of the node set to which distribution factors are to be written.

`int start_node_num` (R)
Location at which to start writing the *node_set_df* for node set ID *node_set_id*.

`int num_df_to_get` (R)
The number of distribution factors to written from *node_set_df*.

`void* node_set_df` (W)
Vector containing the distribution factors for the specified node set are to be written.

2.6 Miscellaneous Functions

These functions are also available via the NEMESIS API.

2.6.1 Read the NEMESIS File Type

The function `ne_get_file_type` reads the file type from a NEMESIS file. Either a *s*, for a scalar load-balance file, or a *p* for a parallel file intended to be read by a single processor. Memory for *ftype* must be allocated prior to calling `ne_get_file_type`.

In case of an error, `ne_get_file_type` returns a negative value.

ne_get_file_type: C Interface

```
int ne_get_file_type(neid, ftype)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`char* ftype` (W)
The type of file associated with neid. Either *s* for a scalar load-balance file, or *p* for a parallel file.

2.6.2 Obtain the Element Type

The function `ne_get_elem_type` obtains the type of element contained in the specified element block. Memory for *elem_type* must be allocate prior to calling `ne_get_elem_type`.

In case of an error, `ne_get_elem_type` returns a negative value.

ne_get_elem_type: C Interface

```
int ne_get_elem_type(neid, elem_blk_id, elem_type)
```

`int neid` (R)
EXODUS II file ID returned from a previous call to `ex_create` or `ex_open`.

`int elem_blk_id` (R)
The element block ID whose element type is to be found.

`char* elem_type` (W)
Pointer to a character array into which the element type is to be stored

A Nodal Based Decomposition Example

A.1 Example Geometry

A simple geometry is shown in Fig. 4 along with the corresponding NEMESIS I / EXODUS II parallel files in Appendix A.2 and A.3, and the scalar load-balance file in Appendix A.4. The files in appendices A.2 through A.4 were obtained with the NetCDF utility *ncdump*. Comments added by the authors are delimited with the "C" convention, i.e., "/* */". Only items which are not found in an EXODUS II database are commented.

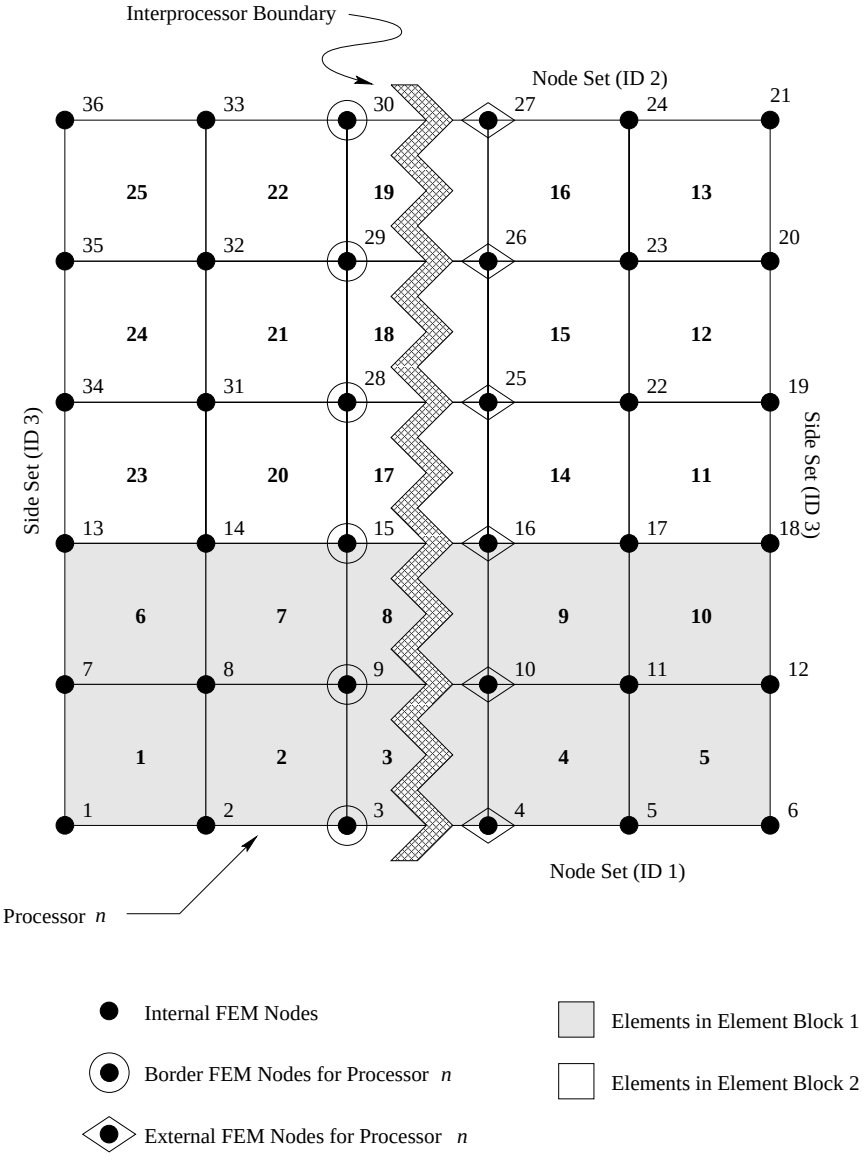

Figure 4: Nodal based decomposition example geometry.

A.2 Processor 0 NEMESIS I / EXODUS II File

```
netcdf testa-m2-bKL.par.2 {
dimensions:
        len_string = 33 ;
        len_line = 81 ;
        four = 4 ;
        time_step = UNLIMITED ; // (0 currently)
        num_nodes_global = 36 ; /* Number of global FEM nodes */
        num_elems_global = 25 ; /* Number of global FEM elements */
        num_el_blk_global = 2 ; /* Number of global element blocks */
        num_ns_global = 2 ;  /* Number of node sets in the GLOBAL FEM mesh */
        num_ss_global = 1 ;  /* Number of side sets in the GLOBAL FEM mesh */
        num_processors = 2 ; /* The number of processors the problem was decomposed for */
        num_procs_file = 1 ; /* The number of processors this file contains info for */
        num_int_elem = 15 ;  /* The number of internal FEM elements on processor 0 */
        num_int_node = 12 ;  /* The number of internal FEM nodes on processor 0 */
        num_bor_node = 6 ;   /* The number of border FEM nodes on processor 0 */
        num_ext_node = 6 ;   /* The number of external FEM nodes on processor 0 */
        num_n_cmaps = 1 ;    /* The number of nodal communication maps for processor 0 */
        ncnt_cmap = 6 ;      /* The count of FEM nodes in communication map 0 on processor 0 */
        num_dim = 2 ;
        num_nodes = 24 ;
        num_elem = 15 ;
        num_el_blk = 2 ;
        num_node_sets = 2 ;
        num_side_sets = 1 ;
        num_qa_rec = 2 ;
        num_el_in_blk1 = 10 ;
        num_nod_per_el1 = 4 ;
        num_el_in_blk2 = 5 ;
        num_nod_per_el2 = 4 ;
        num_nod_ns1 = 6 ;
        num_side_ss1 = 6 ;
        num_df_ss1 = 12 ;
variables:
        float time_whole(time_step) ;
        long el_blk_ids_global(num_el_blk_global) ;
        long ns_ids_global(num_ns_global) ;
        long ns_node_cnt_global(num_ns_global) ;
        long ns_df_cnt_global(num_ns_global) ;
        long ss_ids_global(num_ss_global) ;
        long ss_side_cnt_global(num_ss_global) ;
        long ss_df_cnt_global(num_ss_global) ;
        long nem_ftype ;
        long int_n_stat(num_procs_file) ;
        long bor_n_stat(num_procs_file) ;
        long ext_n_stat(num_procs_file) ;
        long int_e_stat(num_procs_file) ;
        long bor_e_stat(num_procs_file) ;
        long elem_mapi(num_int_elem) ;
        long node_mapi(num_int_node) ;
        long node_mapb(num_bor_node) ;
        long node_mape(num_ext_node) ;
        long n_comm_ids(num_n_cmaps) ;
        long n_comm_stat(num_n_cmaps) ;
        long n_comm_data_idx(num_n_cmaps) ;
```

```
        long n_comm_nids(ncnt_cmap) ;
        long n_comm_proc(ncnt_cmap) ;
        long eb_status(num_el_blk) ;
        long eb_prop1(num_el_blk) ;
                eb_prop1:name = "ID" ;
        long ns_status(num_node_sets) ;
        long ns_prop1(num_node_sets) ;
                ns_prop1:name = "ID" ;
        long ss_status(num_side_sets) ;
        long ss_prop1(num_side_sets) ;
                ss_prop1:name = "ID" ;
        float coord(num_dim, num_nodes) ;
        char coor_names(num_dim, len_string) ;
        char qa_records(num_qa_rec, four, len_string) ;
        long node_num_map(num_nodes) ;
        long elem_num_map(num_elem) ;
        long connect1(num_el_in_blk1, num_nod_per_el1) ;
                connect1:elem_type = "QUAD    " ;
        long connect2(num_el_in_blk2, num_nod_per_el2) ;
                connect2:elem_type = "QUAD    " ;
        long node_ns1(num_nod_ns1) ;
        float dist_fact_ns1(num_nod_ns1) ;
        long elem_ss1(num_side_ss1) ;
        long side_ss1(num_side_ss1) ;
        float dist_fact_ss1(num_df_ss1) ;

// global attributes:
                :api_version = 2.0599999f ;
                :version = 2.02f ;
                :floating_point_word_size = 4 ;
                :nemesis_file_version = 2.5f ;  /* Nemesis file version information */
                :nemesis_api_version = 2.f ;    /* Nemesis API version information */
                :title = "" ;

data:

 el_blk_ids_global = 1, 2 ;     /* Vector of global element block IDs */

 ns_ids_global = 1, 2 ;         /* Vector of global node set IDs */

 ns_node_cnt_global = 6, 6 ;    /* Vector of global node counts in each global node set */

 ns_df_cnt_global = 0, 0 ;      /* Vector of dist. factor counts in each global node set */

 ss_ids_global = 3 ;            /* Vector of global side set IDs */

 ss_side_cnt_global = 10 ;      /* Vector of global side counts in each global side set */

 ss_df_cnt_global = 0 ;         /* Vector of dist. factor counts in each global side set */

 nem_ftype = 0 ;                /* The type of Nemesis~I file */

 int_n_stat = 1 ;               /* The status vector for the internal nodes */

 bor_n_stat = 1 ;               /* The status vector for the border nodes */

 ext_n_stat = 1 ;               /* The status vector for the external nodes */
```

```
int_e_stat = 1 ;              /* The status vector for the internal elements */

bor_e_stat = 0 ;              /* The status vector for the border elements */

elem_mapi = 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15; /* Vector of internal element IDs */

node_mapi = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 ;     /* Vector of internal node IDs */

node_mapb = 13, 14, 15, 16, 17, 18 ;                       /* Vector of border node IDs */

node_mape = 19, 20, 21, 22, 23, 24 ;                       /* Vector of external node IDs */

n_comm_ids = 1 ;              /* Vector of nodal communication map IDs */

n_comm_stat = 1 ;             /* Status vector for nodal communication maps */

n_comm_data_idx = 6 ;         /* Index into nodal communication map */

n_comm_nids = 19, 20, 21, 22, 23, 24 ;     /* Nodal IDs in nodal communication maps */

n_comm_proc = 1, 1, 1, 1, 1, 1 ; /* Processor IDs for each node in nodal communication maps */

eb_status = 1, 1 ;

eb_prop1 = 1, 2 ;

ns_status = 1, 0 ;

ns_prop1 = 1, 2 ;

ss_status = 1 ;

ss_prop1 = 3 ;

coord =
 0, 0.2, 0.4, 0.6, 0.8, 1, 0, 0.2, 0.4000001, 0.6, 0.8, 1, 0, 0.2, 0.4, 0.6,
   0.8, 1, 1, 0.8, 0.6, 0.4, 0.2, 0,
 0, 0, 0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.4, 0.4, 0.4, 0.4,
   0.4, 0.6, 0.6, 0.5999999, 0.5999999, 0.5999999, 0.5999999 ;

coor_names =
 "X",
 "Y" ;

qa_records =
 "FASTQ",
 " 2.1",
 "12/09/92",
 "13:39:32",
 "nem_spread",
 "2.82",
 "23 Jul 1997",
 "10:16:38" ;

node_num_map = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
   18, 19, 22, 25, 28, 31, 34 ;
```

```
 elem_num_map = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 17, 20, 23 ;

 connect1 =
  1, 2, 8, 7,
  2, 3, 9, 8,
  3, 4, 10, 9,
  4, 5, 11, 10,
  5, 6, 12, 11,
  7, 8, 14, 13,
  8, 9, 15, 14,
  9, 10, 16, 15,
  10, 11, 17, 16,
  11, 12, 18, 17 ;

 connect2 =
  18, 19, 20, 17,
  17, 20, 21, 16,
  16, 21, 22, 15,
  15, 22, 23, 14,
  14, 23, 24, 13 ;

 node_ns1 = 1, 2, 3, 4, 5, 6 ;

 dist_fact_ns1 = 1, 1, 1, 1, 1, 1 ;

 elem_ss1 = 1, 5, 6, 10, 11, 15 ;

 side_ss1 = 4, 2, 4, 2, 1, 3 ;

 dist_fact_ss1 = 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ;
}
```

A.3 Processor 1 NEMESIS I / EXODUS II File

```
netcdf testa-m2-bKL.par.2 {
dimensions:
        len_string = 33 ;
        len_line = 81 ;
        four = 4 ;
        time_step = UNLIMITED ; // (0 currently)
        num_nodes_global = 36 ;
        num_elems_global = 25 ;
        num_el_blk_global = 2 ;
        num_ns_global = 2 ;
        num_ss_global = 1 ;
        num_processors = 2 ;
        num_procs_file = 1 ;
        num_int_elem = 15 ;
        num_int_node = 12 ;
        num_bor_node = 6 ;
        num_ext_node = 6 ;
        num_n_cmaps = 1 ;
        ncnt_cmap = 6 ;
        num_dim = 2 ;
        num_nodes = 24 ;
        num_elem = 15 ;
```

```
        num_el_blk = 2 ;
        num_node_sets = 2 ;
        num_side_sets = 1 ;
        num_qa_rec = 1 ;
        num_el_in_blk2 = 15 ;
        num_nod_per_el2 = 4 ;
        num_nod_ns2 = 6 ;
        num_side_ss1 = 6 ;
        num_df_ss1 = 12 ;
variables:
        float time_whole(time_step) ;
        long el_blk_ids_global(num_el_blk_global) ;
        long ns_ids_global(num_ns_global) ;
        long ns_node_cnt_global(num_ns_global) ;
        long ns_df_cnt_global(num_ns_global) ;
        long ss_ids_global(num_ss_global) ;
        long ss_side_cnt_global(num_ss_global) ;
        long ss_df_cnt_global(num_ss_global) ;
        long nem_ftype ;
        long int_n_stat(num_procs_file) ;
        long bor_n_stat(num_procs_file) ;
        long ext_n_stat(num_procs_file) ;
        long int_e_stat(num_procs_file) ;
        long bor_e_stat(num_procs_file) ;
        long elem_mapi(num_int_elem) ;
        long node_mapi(num_int_node) ;
        long node_mapb(num_bor_node) ;
        long node_mape(num_ext_node) ;
        long n_comm_ids(num_n_cmaps) ;
        long n_comm_stat(num_n_cmaps) ;
        long n_comm_data_idx(num_n_cmaps) ;
        long n_comm_nids(ncnt_cmap) ;
        long n_comm_proc(ncnt_cmap) ;
        long eb_status(num_el_blk) ;
        long eb_prop1(num_el_blk) ;
                eb_prop1:name = "ID" ;
        long ns_status(num_node_sets) ;
        long ns_prop1(num_node_sets) ;
                ns_prop1:name = "ID" ;
        long ss_status(num_side_sets) ;
        long ss_prop1(num_side_sets) ;
                ss_prop1:name = "ID" ;
        float coord(num_dim, num_nodes) ;
        char coor_names(num_dim, len_string) ;
        char qa_records(num_qa_rec, four, len_string) ;
        long node_num_map(num_nodes) ;
        long elem_num_map(num_elem) ;
        long connect2(num_el_in_blk2, num_nod_per_el2) ;
                connect2:elem_type = "QUAD    " ;
        long node_ns2(num_nod_ns2) ;
        float dist_fact_ns2(num_nod_ns2) ;
        long elem_ss1(num_side_ss1) ;
        long side_ss1(num_side_ss1) ;
        float dist_fact_ss1(num_df_ss1) ;

// global attributes:
                :api_version = 2.0599999f ;
```

```
                :version = 2.02f ;
                :floating_point_word_size = 4 ;
                :nemesis_file_version = 2.5f ;
                :nemesis_api_version = 2.f ;
                :title = "Parallel Mesh File for Processor 1" ;

data:

 el_blk_ids_global = 0, 0 ;

 ns_ids_global = 0, 0 ;

 ns_node_cnt_global = 0, 0 ;

 ns_df_cnt_global = 0, 0 ;

 ss_ids_global = 0 ;

 ss_side_cnt_global = 0 ;

 ss_df_cnt_global = 0 ;

 nem_ftype = 0 ;

 int_n_stat = 1 ;

 bor_n_stat = 1 ;

 ext_n_stat = 1 ;

 int_e_stat = 1 ;

 bor_e_stat = 0 ;

 elem_mapi = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ;

 node_mapi = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 ;

 node_mapb = 13, 14, 15, 16, 17, 18 ;

 node_mape = 19, 20, 21, 22, 23, 24 ;

 n_comm_ids = 0 ;

 n_comm_stat = 1 ;

 n_comm_data_idx = 6 ;

 n_comm_nids = 19, 20, 21, 22, 23, 24 ;

 n_comm_proc = 0, 0, 0, 0, 0, 0 ;

 eb_status = 0, 1 ;

 eb_prop1 = 1, 2 ;

 ns_status = 0, 1 ;
```

```
 ns_prop1 = 1, 2 ;

 ss_status = 1 ;

 ss_prop1 = 3 ;

 coord =
  1, 1, 0.7999999, 0.8, 0.6, 0.6, 0.4, 0.4, 0.2, 0.2, 0, 0, 1, 0.8, 0.6, 0.4,
    0.2, 0, 0, 0.2, 0.4, 0.6, 0.8, 1,
  0.8000001, 1, 0.8000001, 1, 0.8000001, 1, 0.8, 1, 0.8, 1, 0.8, 1, 0.6, 0.6,
    0.5999999, 0.5999999, 0.5999999, 0.5999999, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4 ;

 coor_names =
  "",
  "" ;

 qa_records =
  "nem_spread",
  "2.82",
  "23 Jul 1997",
  "10:16:40" ;

 node_num_map = 20, 21, 23, 24, 26, 27, 29, 30, 32, 33, 35, 36, 19, 22, 25,
    28, 31, 34, 13, 14, 15, 16, 17, 18 ;

 elem_num_map = 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25 ;

 connect2 =
  24, 13, 14, 23,
  13, 1, 3, 14,
  1, 2, 4, 3,
  23, 14, 15, 22,
  14, 3, 5, 15,
  3, 4, 6, 5,
  22, 15, 16, 21,
  15, 5, 7, 16,
  5, 6, 8, 7,
  21, 16, 17, 20,
  16, 7, 9, 17,
  7, 8, 10, 9,
  20, 17, 18, 19,
  17, 9, 11, 18,
  9, 10, 12, 11 ;

 node_ns2 = 2, 4, 6, 8, 10, 12 ;

 dist_fact_ns2 = 1, 1, 1, 1, 1, 1 ;

 elem_ss1 = 1, 2, 3, 13, 14, 15 ;

 side_ss1 = 1, 1, 1, 3, 3, 3 ;

 dist_fact_ss1 = 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ;
}
```

A.4 Scalar Load-Balance File

```
netcdf testa-m2-bKL {
dimensions:
        len_string = 33 ;
        len_line = 81 ;
        four = 4 ;
        time_step = UNLIMITED ; // (0 currently)
        num_info = 3 ;
        num_nodes_global = 36 ;   /* Number of global FEM nodes */
        num_elems_global = 25 ;   /* Number of global FEM elements */
        num_el_blk_global = 2 ;   /* Number of global element blocks */
        num_processors = 2 ;      /* Number of processors for which the geometry was decomposed */
        num_procs_file = 2 ;      /* Number of processors this file contains information for */
        num_qa_rec = 1 ;
        num_int_elem = 30 ;       /* Number of internal elements (for all processors) */
        num_int_node = 24 ;       /* Number of internal nodes (for all processors)  */
        num_bor_node = 12 ;       /* Number of border nodes (for all processors) */
        num_ext_node = 12 ;       /* Number of external nodes (for all processors) */
        num_n_cmaps = 2 ;         /* Number of nodal communication maps (for all processors) */
        ncnt_cmap = 12 ;  /* Node count in nodal comm. maps (all maps, all processors) */
variables:
        float time_whole(time_step) ;
        char info_records(num_info, len_line) ;
        long el_blk_ids_global(num_el_blk_global) ;
        long nem_ftype ;
        char qa_records(num_qa_rec, four, len_string) ;
        long int_n_stat(num_procs_file) ;
        long bor_n_stat(num_procs_file) ;
        long ext_n_stat(num_procs_file) ;
        long int_e_stat(num_procs_file) ;
        long bor_e_stat(num_procs_file) ;
        long elem_mapi(num_int_elem) ;
        long elem_mapi_idx(num_procs_file) ;
        long node_mapi(num_int_node) ;
        long node_mapi_idx(num_procs_file) ;
        long node_mapb(num_bor_node) ;
        long node_mapb_idx(num_procs_file) ;
        long node_mape(num_ext_node) ;
        long node_mape_idx(num_procs_file) ;
        long n_comm_ids(num_n_cmaps) ;
        long n_comm_stat(num_n_cmaps) ;
        long n_comm_info_idx(num_procs_file) ;
        long n_comm_data_idx(num_n_cmaps) ;
        long n_comm_nids(ncnt_cmap) ;
        long n_comm_proc(ncnt_cmap) ;

// global attributes:
                :api_version = 2.0599999f ;
                :version = 2.02f ;
                :floating_point_word_size = 4 ;
                :nemesis_file_version = 2.5f ;  /* Nemesis file version information */
                :nemesis_api_version = 2.f ;    /* Nemesis API version information */

data:
 info_records =
  "nem_slice nodal load balance file",
```

```
 "method1: Multilevel-KL decomposition via bisection",
 "method2: With Kernighan-Lin refinement" ;

el_blk_ids_global = 0, 0 ;

nem_ftype = 1 ;

qa_records =
 "nem_slice",
 "2.1",
 "23 Jul 1997",
 "10:16:30" ;

int_n_stat = 1, 1 ;    /* The status vector for internal node vectors on each processor */

bor_n_stat = 1, 1 ;    /* The status vector for border node vectors on each processor */

ext_n_stat = 1, 1 ;    /* The status vector for external node vectors on each processor */

int_e_stat = 1, 1 ;    /* The status vector for internal element vectors on each processor */

bor_e_stat = 0, 0 ;    /* The status vector for border element vectors on each processor */

 /* Information and index vectors for the processors */
 /* each vector contains the information for all of the processors */
 /* the index vectors, denoted by the _idx, gives the starting     */
 /* position in the information vector for each processor          */
elem_mapi = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 16, 19, 22, 10, 11, 12,
   13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24 ;

elem_mapi_idx = 15, 30 ;

node_mapi = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 19, 20, 22, 23, 25, 26,
   28, 29, 31, 32, 34, 35 ;

node_mapi_idx = 12, 24 ;

node_mapb = 12, 13, 14, 15, 16, 17, 18, 21, 24, 27, 30, 33 ;

node_mapb_idx = 6, 12 ;

node_mape = 18, 21, 24, 27, 30, 33, 12, 13, 14, 15, 16, 17 ;

node_mape_idx = 6, 12 ;

n_comm_ids = 1, 1 ;

n_comm_stat = 1, 1 ;

n_comm_info_idx = 1, 2 ;

n_comm_data_idx = 6, 12 ;

n_comm_nids = 18, 21, 24, 27, 30, 33, 12, 13, 14, 15, 16, 17 ;

n_comm_proc = 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 ;
}
```

B Elemental Based Decomposition Example

B.1 Example Geometry

A simple geometry is shown in Fig. 5 along with the corresponding NEMESIS I / EXODUS II parallel files in Appendix B.2 and B.3, and the scalar load-balance file in Appendix B.4. The files in appendices B.2 through B.4 were obtained with the NetCDF utility *ncdump*. Comments added by the authors are delimited with the "C" convention, i.e., "/* */". Only items which are not found in an EXODUS II database are commented.

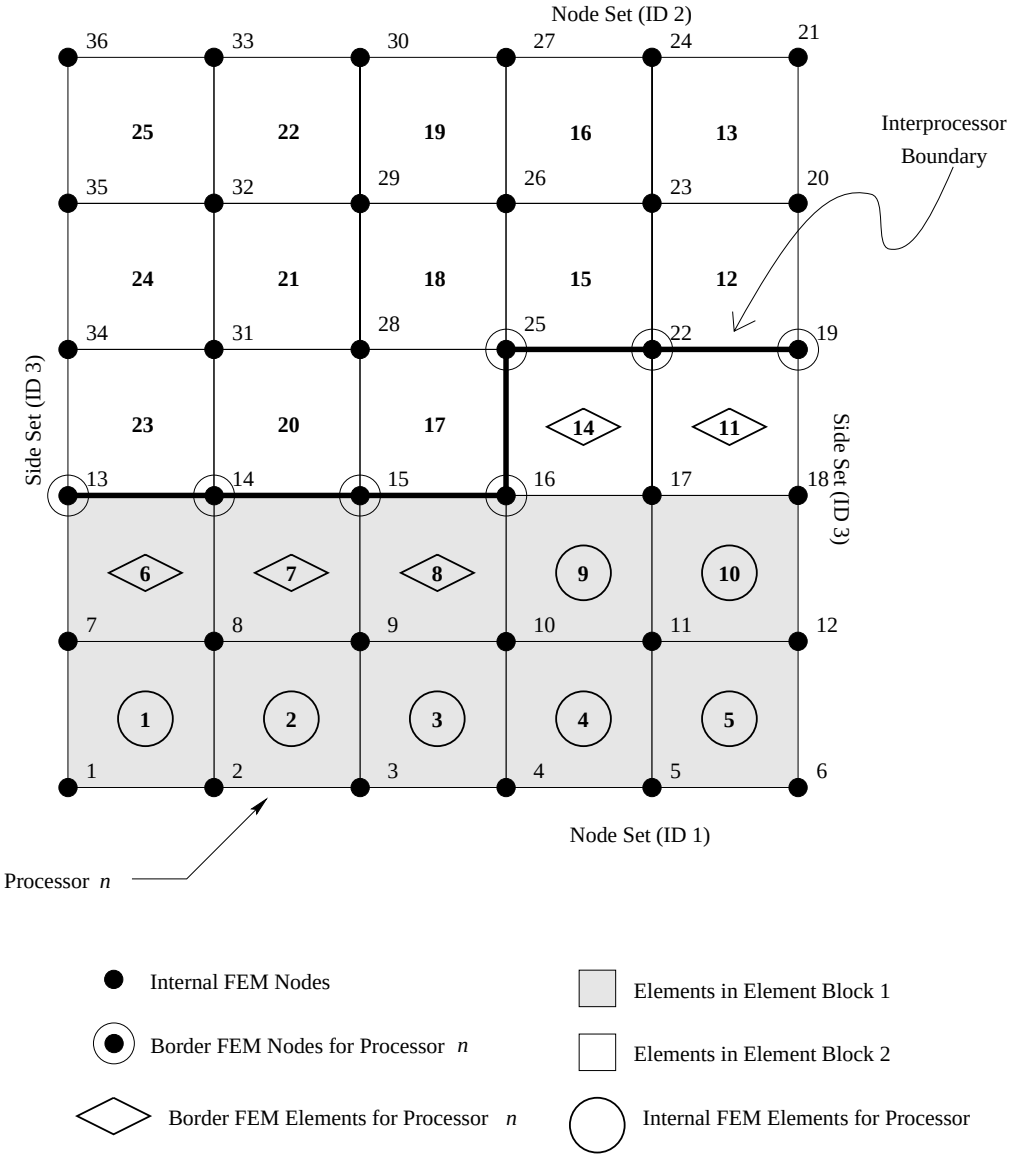

Figure 5: Elemental based decomposition example geometry.

B.2 Processor 0 NEMESIS I / EXODUS II File

```
netcdf testa-m2-bKL.par.2 {
dimensions:
        len_string = 33 ;
        len_line = 81 ;
        four = 4 ;
        time_step = UNLIMITED ; // (0 currently)
        num_nodes_global = 36 ; /* Number of global FEM nodes */
        num_elems_global = 25 ; /* Number of global FEM elements */
        num_el_blk_global = 2 ; /* Number of global element blocks */
        num_ns_global = 2 ;  /* Number of node sets in the GLOBAL FEM mesh */
        num_ss_global = 1 ;  /* Number of side sets in the GLOBAL FEM mesh */
        num_processors = 2 ; /* The number of processors the problem was decomposed for */
        num_procs_file = 1 ; /* The number of processors this file contains info for */
        num_int_elem = 7 ;   /* The number of internal FEM elements on processor 0 */
        num_bor_elem = 5 ;   /* The number of border FEM elements on processor 0 */
        num_int_node = 14 ;  /* The number of internal FEM nodes on processor 0 */
        num_bor_node = 7 ;   /* The number of border FEM nodes on processor 0 */
        num_n_cmaps = 1 ;    /* The number of nodal communication maps for processor 0 */
        num_e_cmaps = 1 ;    /* The number of elemental communication maps for processor 0 */
        ncnt_cmap = 7 ;      /* The count of FEM nodes in communication map 0 on processor 0 */
        ecnt_cmap = 6 ;      /* The count of FEM elements in communication map 0 on processor 0 */
        num_dim = 2 ;
        num_nodes = 21 ;
        num_elem = 12 ;
        num_el_blk = 2 ;
        num_node_sets = 2 ;
        num_side_sets = 1 ;
        num_qa_rec = 2 ;
        num_el_in_blk1 = 10 ;
        num_nod_per_el1 = 4 ;
        num_el_in_blk2 = 2 ;
        num_nod_per_el2 = 4 ;
        num_nod_ns1 = 6 ;
        num_side_ss1 = 5 ;
        num_df_ss1 = 10 ;
variables:
        float time_whole(time_step) ;
        long el_blk_ids_global(num_el_blk_global) ;
        long ns_ids_global(num_ns_global) ;
        long ns_node_cnt_global(num_ns_global) ;
        long ns_df_cnt_global(num_ns_global) ;
        long ss_ids_global(num_ss_global) ;
        long ss_side_cnt_global(num_ss_global) ;
        long ss_df_cnt_global(num_ss_global) ;
        long nem_ftype ;
        long int_n_stat(num_procs_file) ;
        long bor_n_stat(num_procs_file) ;
        long ext_n_stat(num_procs_file) ;
        long int_e_stat(num_procs_file) ;
        long bor_e_stat(num_procs_file) ;
        long elem_mapi(num_int_elem) ;
        long elem_mapb(num_bor_elem) ;
        long node_mapi(num_int_node) ;
        long node_mapb(num_bor_node) ;
        long n_comm_ids(num_n_cmaps) ;
```

```
        long n_comm_stat(num_n_cmaps) ;
        long e_comm_ids(num_e_cmaps) ;
        long e_comm_stat(num_e_cmaps) ;
        long n_comm_data_idx(num_n_cmaps) ;
        long n_comm_nids(ncnt_cmap) ;
        long n_comm_proc(ncnt_cmap) ;
        long e_comm_data_idx(num_e_cmaps) ;
        long e_comm_eids(ecnt_cmap) ;
        long e_comm_proc(ecnt_cmap) ;
        long e_comm_sids(ecnt_cmap) ;
        long eb_status(num_el_blk) ;
        long eb_prop1(num_el_blk) ;
                eb_prop1:name = "ID" ;
        long ns_status(num_node_sets) ;
        long ns_prop1(num_node_sets) ;
                ns_prop1:name = "ID" ;
        long ss_status(num_side_sets) ;
        long ss_prop1(num_side_sets) ;
                ss_prop1:name = "ID" ;
        float coord(num_dim, num_nodes) ;
        char coor_names(num_dim, len_string) ;
        char qa_records(num_qa_rec, four, len_string) ;
        long node_num_map(num_nodes) ;
        long elem_num_map(num_elem) ;
        long connect1(num_el_in_blk1, num_nod_per_el1) ;
                connect1:elem_type = "QUAD    " ;
        long connect2(num_el_in_blk2, num_nod_per_el2) ;
                connect2:elem_type = "QUAD    " ;
        long node_ns1(num_nod_ns1) ;
        float dist_fact_ns1(num_nod_ns1) ;
        long elem_ss1(num_side_ss1) ;
        long side_ss1(num_side_ss1) ;
        float dist_fact_ss1(num_df_ss1) ;

// global attributes:
                :api_version = 2.0599999f ;
                :version = 2.02f ;
                :floating_point_word_size = 4 ;
                :nemesis_file_version = 2.5f ;  /* Nemesis file version information */
                :nemesis_api_version = 2.f ;    /* Nemesis API version information */
                :title = "" ;

data:

 el_blk_ids_global = 1, 2 ;     /* Vector of global element block IDs */

 ns_ids_global = 1, 2 ;         /* Vector of global node set IDs */

 ns_node_cnt_global = 6, 6 ;    /* Vector of global node counts in each global node set */

 ns_df_cnt_global = 0, 0 ;      /* Vector of dist. factor counts in each global node set */

 ss_ids_global = 3 ;            /* Vector of global side set IDs */

 ss_side_cnt_global = 10 ;      /* Vector of global side counts in each global side set */

 ss_df_cnt_global = 0 ;         /* Vector of dist. factor counts in each global side set */
```

```
nem_ftype = 0 ;                /* The type of Nemesis~I file */

int_n_stat = 1 ;               /* The status vector for the internal nodes */

bor_n_stat = 1 ;               /* The status vector for the border nodes */

ext_n_stat = 0 ;               /* The status vector for the external nodes */

int_e_stat = 1 ;               /* The status vector for the internal elements */

bor_e_stat = 1 ;               /* The status vector for the border elements */

elem_mapi = 1, 2, 3, 4, 5, 9, 10 ;     /* Vector of internal element IDs */

elem_mapb = 6, 7, 8, 11, 12 ;          /* Vector of border element IDs */

node_mapi = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 ;    /* Vector of internal node IDs */

node_mapb = 15, 16, 17, 18, 19, 20, 21 ;                       /* Vector of border node IDs */

n_comm_ids = 1 ;               /* Vector of nodal communication map IDs */

n_comm_stat = 1 ;              /* Status vector for nodal communication maps */

e_comm_ids = 1 ;               /* Vector of elemental communication map IDs */

e_comm_stat = 1 ;              /* Status vector for elemental communication maps */

n_comm_data_idx = 7 ;          /* Index into nodal communication map */

n_comm_nids = 15, 16, 17, 18, 19, 20, 21 ;     /* Nodal IDs in nodal communication maps */

n_comm_proc = 1, 1, 1, 1, 1, 1, 1 ; /* Processor IDs for each node in nodal communication maps */

e_comm_data_idx = 6 ;          /* Index into elemental communication map */

e_comm_eids = 6, 7, 8, 11, 12, 12 ;    /* Elemental IDs in elemental communication maps */

e_comm_proc = 1, 1, 1, 1, 1, 1 ; /* Processor IDs for each element in elemental comm. maps */

e_comm_sids = 3, 3, 3, 2, 3, 2 ;   /* The side IDs of sides shared between processors */

eb_status = 1, 1 ;

eb_prop1 = 1, 2 ;

ns_status = 1, 0 ;

ns_prop1 = 1, 2 ;

ss_status = 1 ;

ss_prop1 = 3 ;

coord =
 0, 0.2, 0.4, 0.6, 0.8, 1, 0, 0.2, 0.4000001, 0.6, 0.8, 1, 0.8, 1, 0, 0.2,
```

```
   0.4, 0.6, 1, 0.8, 0.6,
 0, 0, 0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.4, 0.4, 0.4, 0.4,
   0.4, 0.6, 0.6, 0.5999999 ;

 coor_names =
  "X",
  "Y" ;

 qa_records =
  "FASTQ",
  " 2.1",
  "12/09/92",
  "13:39:32",
  "nem_spread",
  "2.82",
  "23 Jul 1997",
  "09:26:38" ;

 node_num_map = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 17, 18, 13, 14, 15,
   16, 19, 22, 25 ;

 elem_num_map = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14 ;

 connect1 =
  1, 2, 8, 7,
  2, 3, 9, 8,
  3, 4, 10, 9,
  4, 5, 11, 10,
  5, 6, 12, 11,
  7, 8, 16, 15,
  8, 9, 17, 16,
  9, 10, 18, 17,
  10, 11, 13, 18,
  11, 12, 14, 13 ;

 connect2 =
  14, 19, 20, 13,
  13, 20, 21, 18 ;

 node_ns1 = 1, 2, 3, 4, 5, 6 ;

 dist_fact_ns1 = 1, 1, 1, 1, 1, 1 ;

 elem_ss1 = 1, 5, 6, 10, 11 ;

 side_ss1 = 4, 2, 4, 2, 1 ;

 dist_fact_ss1 = 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ;
}
```

B.3 Processor 1 NEMESIS I / EXODUS II File

```
netcdf testa-m2-bKL.par.2 {
dimensions:
        len_string = 33 ;
        len_line = 81 ;
        four = 4 ;
```

```
        time_step = UNLIMITED ; // (0 currently)
        num_nodes_global = 36 ;
        num_elems_global = 25 ;
        num_el_blk_global = 2 ;
        num_ns_global = 2 ;
        num_ss_global = 1 ;
        num_processors = 2 ;
        num_procs_file = 1 ;
        num_int_elem = 8 ;
        num_bor_elem = 5 ;
        num_int_node = 15 ;
        num_bor_node = 7 ;
        num_n_cmaps = 1 ;
        num_e_cmaps = 1 ;
        ncnt_cmap = 7 ;
        ecnt_cmap = 6 ;
        num_dim = 2 ;
        num_nodes = 22 ;
        num_elem = 13 ;
        num_el_blk = 2 ;
        num_node_sets = 2 ;
        num_side_sets = 1 ;
        num_qa_rec = 1 ;
        num_el_in_blk2 = 13 ;
        num_nod_per_el2 = 4 ;
        num_nod_ns2 = 6 ;
        num_side_ss1 = 5 ;
        num_df_ss1 = 10 ;
variables:
        float time_whole(time_step) ;
        long el_blk_ids_global(num_el_blk_global) ;
        long el_blk_cnt_global(num_el_blk_global) ;
        long ns_ids_global(num_ns_global) ;
        long ns_node_cnt_global(num_ns_global) ;
        long ns_df_cnt_global(num_ns_global) ;
        long ss_ids_global(num_ss_global) ;
        long ss_side_cnt_global(num_ss_global) ;
        long ss_df_cnt_global(num_ss_global) ;
        long nem_ftype ;
        long int_n_stat(num_procs_file) ;
        long bor_n_stat(num_procs_file) ;
        long ext_n_stat(num_procs_file) ;
        long int_e_stat(num_procs_file) ;
        long bor_e_stat(num_procs_file) ;
        long elem_mapi(num_int_elem) ;
        long elem_mapb(num_bor_elem) ;
        long node_mapi(num_int_node) ;
        long node_mapb(num_bor_node) ;
        long n_comm_ids(num_n_cmaps) ;
        long n_comm_stat(num_n_cmaps) ;
        long e_comm_ids(num_e_cmaps) ;
        long e_comm_stat(num_e_cmaps) ;
        long n_comm_data_idx(num_n_cmaps) ;
        long n_comm_nids(ncnt_cmap) ;
        long n_comm_proc(ncnt_cmap) ;
        long e_comm_data_idx(num_e_cmaps) ;
        long e_comm_eids(ecnt_cmap) ;
```

```
        long e_comm_proc(ecnt_cmap) ;
        long e_comm_sids(ecnt_cmap) ;
        long eb_status(num_el_blk) ;
        long eb_prop1(num_el_blk) ;
                eb_prop1:name = "ID" ;
        long ns_status(num_node_sets) ;
        long ns_prop1(num_node_sets) ;
                ns_prop1:name = "ID" ;
        long ss_status(num_side_sets) ;
        long ss_prop1(num_side_sets) ;
                ss_prop1:name = "ID" ;
        float coord(num_dim, num_nodes) ;
        char coor_names(num_dim, len_string) ;
        char qa_records(num_qa_rec, four, len_string) ;
        long node_num_map(num_nodes) ;
        long elem_num_map(num_elem) ;
        long connect2(num_el_in_blk2, num_nod_per_el2) ;
                connect2:elem_type = "QUAD    " ;
        long node_ns2(num_nod_ns2) ;
        float dist_fact_ns2(num_nod_ns2) ;
        long elem_ss1(num_side_ss1) ;
        long side_ss1(num_side_ss1) ;
        float dist_fact_ss1(num_df_ss1) ;

// global attributes:
                :api_version = 2.0599999f ;
                :version = 2.02f ;
                :floating_point_word_size = 4 ;
                :nemesis_file_version = 2.5f ;
                :nemesis_api_version = 2.f ;
                :title = "Parallel Mesh File for Processor 1" ;

data:

 el_blk_ids_global = 1, 2 ;

 el_blk_cnt_global = 10, 15 ;

 ns_ids_global = 1, 2 ;

 ns_node_cnt_global = 6, 6 ;

 ns_df_cnt_global = 0, 0 ;

 ss_ids_global = 3 ;

 ss_side_cnt_global = 10 ;

 ss_df_cnt_global = 0 ;

 nem_ftype = 0 ;

 int_n_stat = 1 ;

 bor_n_stat = 1 ;

 ext_n_stat = 0 ;
```

```
int_e_stat = 1 ;

bor_e_stat = 1 ;

elem_mapi = 2, 4, 6, 7, 9, 10, 12, 13 ;

elem_mapb = 1, 3, 5, 8, 11 ;

node_mapi = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ;

node_mapb = 16, 17, 18, 19, 20, 21, 22 ;

n_comm_ids = 0 ;

n_comm_stat = 1 ;

e_comm_ids = 0 ;

e_comm_stat = 1 ;

n_comm_data_idx = 7 ;

n_comm_nids = 16, 17, 18, 19, 20, 21, 22 ;

n_comm_proc = 0, 0, 0, 0, 0, 0, 0 ;

e_comm_data_idx = 6 ;

e_comm_eids = 11, 8, 5, 1, 5, 3 ;

e_comm_proc = 0, 0, 0, 0, 0, 0 ;

e_comm_sids = 4, 4, 4, 4, 1, 4 ;

eb_status = 0, 1 ;

eb_prop1 = 1, 2 ;

ns_status = 0, 1 ;

ns_prop1 = 1, 2 ;

ss_status = 1 ;

ss_prop1 = 3 ;

coord =
 1, 1, 0.7999999, 0.8, 0.6, 0.6, 0.4, 0.4, 0.4, 0.2, 0.2, 0.2, 0, 0, 0, 0,
   0.2, 0.4, 0.6, 1, 0.8, 0.6,
 0.8000001, 1, 0.8000001, 1, 0.8000001, 1, 0.5999999, 0.8, 1, 0.5999999,
   0.8, 1, 0.5999999, 0.8, 1, 0.4, 0.4, 0.4, 0.4, 0.6, 0.6, 0.5999999 ;

coor_names =
 "",
 "" ;
```

```
 qa_records =
  "nem_spread",
  "2.82",
  "23 Jul 1997",
  "10:31:47" ;

 node_num_map = 20, 21, 23, 24, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
    13, 14, 15, 16, 19, 22, 25 ;

 elem_num_map = 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25 ;

 connect2 =
  20, 1, 3, 21,
  1, 2, 4, 3,
  21, 3, 5, 22,
  3, 4, 6, 5,
  19, 22, 7, 18,
  22, 5, 8, 7,
  5, 6, 9, 8,
  18, 7, 10, 17,
  7, 8, 11, 10,
  8, 9, 12, 11,
  17, 10, 13, 16,
  10, 11, 14, 13,
  11, 12, 15, 14 ;

 node_ns2 = 2, 4, 6, 9, 12, 15 ;

 dist_fact_ns2 = 1, 1, 1, 1, 1, 1 ;

 elem_ss1 = 1, 2, 11, 12, 13 ;

 side_ss1 = 1, 1, 3, 3, 3 ;

 dist_fact_ss1 = 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ;
}
```

B.4 Scalar Load-Balance File

```
netcdf testa-m2-bKL {
dimensions:
        len_string = 33 ;
        len_line = 81 ;
        four = 4 ;
        time_step = UNLIMITED ; // (0 currently)
        num_info = 3 ;
        num_nodes_global = 36 ;   /* Number of global FEM nodes */
        num_elems_global = 25 ;   /* Number of global FEM elements */
        num_el_blk_global = 2 ;   /* Number of global element blocks */
        num_processors = 2 ;      /* Number of processors for which the geometry was decomposed */
        num_procs_file = 2 ;      /* Number of processors this file contains information for */
        num_qa_rec = 1 ;
        num_int_elem = 15 ;       /* Number of internal elements (for all processors) */
        num_bor_elem = 10 ;       /* Number of border elements (for all processors) */
        num_int_node = 29 ;       /* Number of internal nodes (for all processors)  */
        num_bor_node = 14 ;       /* Number of border nodes (for all processors) */
        num_n_cmaps = 2 ;         /* Number of nodal communication maps (for all processors) */
        num_e_cmaps = 2 ;         /* Number of elemental communication maps (for all processors) */
        ncnt_cmap = 14 ;  /* Node count in nodal comm. maps (all maps, all processors) */
        ecnt_cmap = 12 ;  /* Element count in elemental comm. maps (all maps, all processors) */
variables:
        float time_whole(time_step) ;
        char info_records(num_info, len_line) ;
        long el_blk_ids_global(num_el_blk_global) ;
        long nem_ftype ;
        char qa_records(num_qa_rec, four, len_string) ;
        long int_n_stat(num_procs_file) ;
        long bor_n_stat(num_procs_file) ;
        long ext_n_stat(num_procs_file) ;
        long int_e_stat(num_procs_file) ;
        long bor_e_stat(num_procs_file) ;
        long elem_mapi(num_int_elem) ;
        long elem_mapi_idx(num_procs_file) ;
        long elem_mapb(num_bor_elem) ;
        long elem_mapb_idx(num_procs_file) ;
        long node_mapi(num_int_node) ;
        long node_mapi_idx(num_procs_file) ;
        long node_mapb(num_bor_node) ;
        long node_mapb_idx(num_procs_file) ;
        long n_comm_ids(num_n_cmaps) ;
        long n_comm_stat(num_n_cmaps) ;
        long n_comm_info_idx(num_procs_file) ;
        long e_comm_ids(num_e_cmaps) ;
        long e_comm_stat(num_e_cmaps) ;
        long e_comm_info_idx(num_procs_file) ;
        long n_comm_data_idx(num_n_cmaps) ;
        long n_comm_nids(ncnt_cmap) ;
        long n_comm_proc(ncnt_cmap) ;
        long e_comm_data_idx(num_e_cmaps) ;
        long e_comm_eids(ecnt_cmap) ;
        long e_comm_proc(ecnt_cmap) ;
        long e_comm_sids(ecnt_cmap) ;

// global attributes:
```

```
                :api_version = 2.0599999f ;
                :version = 2.02f ;
                :floating_point_word_size = 4 ;
                :nemesis_file_version = 2.5f ;  /* Nemesis file version information */
                :nemesis_api_version = 2.f ;    /* Nemesis API version information */

data:

 info_records =
  "nem_slice elemental load balance file",
  "method1: Multilevel-KL decomposition via bisection",
  "method2: With Kernighan-Lin refinement" ;

 el_blk_ids_global = 0, 0 ;

 nem_ftype = 1 ;        /* The Nemesis I file type */

 qa_records =
  "nem_slice",
  "2.1",
  "23 Jul 1997",
  "09:17:03" ;

 int_n_stat = 1, 1 ;    /* The status vector for internal node vectors on each processor */

 bor_n_stat = 1, 1 ;    /* The status vector for border node vectors on each processor */

 ext_n_stat = 0, 0 ;    /* The status vector for external node vectors on each processor */

 int_e_stat = 1, 1 ;    /* The status vector for internal element vectors on each processor */

 bor_e_stat = 1, 1 ;    /* The status vector for border element vectors on each processor */

  /* Information and index vectors for the processors */
  /* each vector contains the information for all of the processors */
  /* the index vectors, denoted by the _idx, gives the starting     */
  /* position in the information vector for each processor          */
 elem_mapi = 0, 1, 2, 3, 4, 8, 9, 12, 15, 17, 18, 20, 21, 23, 24 ;
 elem_mapi_idx = 7, 15 ;

 elem_mapb = 5, 6, 7, 10, 13, 11, 14, 16, 19, 22 ;

 elem_mapb_idx = 5, 10 ;

 node_mapi = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 16, 17, 19, 20, 22, 23,
    25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35 ;

 node_mapi_idx = 14, 29 ;

 node_mapb = 12, 13, 14, 15, 18, 21, 24, 12, 13, 14, 15, 18, 21, 24 ;

 node_mapb_idx = 7, 14 ;

 n_comm_ids = 1, 1 ;

 n_comm_stat = 1, 1 ;
```

```
 n_comm_info_idx = 1, 2 ;

 e_comm_ids = 1, 1 ;

 e_comm_stat = 1, 1 ;

 e_comm_info_idx = 1, 2 ;

 n_comm_data_idx = 7, 14 ;

 n_comm_nids = 12, 13, 14, 15, 18, 21, 24, 12, 13, 14, 15, 18, 21, 24 ;

 n_comm_proc = 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0 ;

 e_comm_data_idx = 6, 12 ;

 e_comm_eids = 5, 6, 7, 10, 13, 13, 22, 19, 16, 11, 16, 14 ;

 e_comm_proc = 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0 ;

 e_comm_sids = 3, 3, 3, 2, 3, 2, 4, 4, 4, 4, 1, 4 ;
}
```