EMaligner Documentation

Release 0.4

Dan Kapner

Aug 23, 2019

Contents:

EMaligner is a scalable linear least squares image stitching and alignment solver that can align millions of images via point correspondences. The solver runs on systems from individual workstations to distributed clusters.

This solver was developed to provide the image alignment steps for [Mahalingam19]. The starting point for this package was the MATLAB-based package described in [KDS18]. This solver is described in [Kapner19].

Compared to that repository, this repository has a number of changes and additions:

	[KDS18]	this repo
language	MATLAB	Python, C for external
external solver	PaStiX	multiple, via PETSc
external solver installation	independent of repo	included in Singularity container
transforms	• translation • rigid approximation • affine • polynomial to 3rd degree	• translation • rigid via rotation transform • affine • polynomial to arbitrary degree • thin plate spline
automated tests	N/A	TravisCI

CHAPTER 1

The User Guide

Use cases and detailed explanations of input parameters.

1.1 User Guide

1.1.1 Use cases

- **montage** (or "stitching" or "registration"): stitching of images in a single section of tissue. Even for a section comprised of thousands of images, these solves can generally be performed on a single workstation or compute node in under a minute.
- **rough alignment**: stitching of downsampled images in 3D. Typically each section is a single downsampled image. Even for thousands of downsampled sections, these solves can generally be performed on a single workstation or compute node in a few minutes. Compared to montages the computation time is increased due to the underlying database structure. Compared to fine alignment, this is just a very small 3D solve.
- **fine alignment**: these solves comprise thousands to millions of images across many z values. The solves are memory-limited and the size of the solve determines whether they can be run on a single node, need distribution across multiple nodes, and, potentially, whether one should choose a direct or iterative distributed solver.

The size of a solve is determined by the number of images, the number of point correspondences derived between the images, and, the number of degrees-of-freedom attributed to each image.

1.1.2 Important Dependencies

- render-python This package provides underlying python interfaces to render
- argschema This package provides the means for setting the many parameters that are inputs to this solver.
- scipy.sparse For single-node solves, this package is used for factorization and solving.
- PETSc This is a large package that supports distributed linear algebra and many different preconditioners and solvers.

- Singularity The compilation and use of PETSc is a steep learning curve. This repo includes a PETSc build and a solver compilation in singularity containers for ease-of-use.

1.1.3 Detailed Argument Descriptions

see *EMaligner schema*

1.1.4 Distributed Usage

EM aligner distributed

For distributed solves across multiple compute nodes. This solver is built with the PETSc libraries

https://www.mcs.anl.gov/petsc/index.html

Systems

This code has been run on NERSC's Cori and on the Allen Insitute cluster.

For Cori, Cray modules make simple work of compiling and running this code. See `makefile_cori` and `cori_example_script`.

For the Allen cluster, one way to run this is via Singularity containers. `Singularity.petsc` is a definition file for an image with compiled PETSc. This is a fairly lengthy process and should not change very often. The build of the image is manually triggered and maintained on Singularity Hub: https://singularity-hub.org/collections/2940. The solver code in this repository is then compiled in another container that builds from the PETSc image. This is `Singularity.petsc_solver`. For an example of how to run this compilation step, look in `.travis.yml` in this repository. For one example of how to use the built singularity image, see `integration_tests/test_hdf5.py`. For another example, see `allen_example_script.pbs`.

Usage

```
em_solver_cori -input <input file> -output <output_file> <ksp options>
```

or

```
singularity run --bind <external>:<internal> em_distributed.simf -input <input_file> -
↪output <output_file> <ksp options>
```

`ksp options` specify how PETSc should handle the system. For example whether to use a direct or iterative solver, what type of preconditioner to use, what external packages to invoke.

- direct solve with Pastix: `-ksp_type preonly -pc_type lu`

There are many PETSc options, and not all of them are necessarily installed in the Singularity image here.

File Formats

For transferring distributed A matrices to the distributed solver. HDFView is a convenient utility for inspecting the contents of hdf5 files. h5py and HDFView sometimes report different object types so we report both here, when necessary.

format of input_file.h5 (output_file.h5 should be a copy with x_0 and x_1 replaced by the new solution)

```
dataset name: datafile_names
    type (h5py):
        object
    type (HDFView):
        String, length = variable, padding = H5T_STR_NULLTERM,
        cset = H5T_CSET_ASCII
    shape:
        (nfile, 1)
    description:
        relative paths of files that contain the distributed A matrix.

dataset name: datafile_maxcol
    type:
        int64
    shape:
        (nfile, 1)
    description:
        maximum column index contained in each file.

dataset name: datafile_mincol
    type:
        int64
    shape:
        (nfile, 1)
    description:
        minimum column index contained in each file.

dataset name: datafile_nnz
    type:
        int64
    shape:
        (nfile, 1)
    description:
        number of nonzeros in each file

dataset name: datafile_nrows
    type:
        int64
    shape:
        (nfile, 1)
    description:
        number of sub-matrix rows in each file

dataset name: input_args
    type (h5py):
        object
    type (HDFView):
        String, length = variable, padding = H5T_STR_NULLTERM,
        cset = H5T_CSET_ASCII
    shape: (1,)
    description:
        copy of the input args dict used as input to the aligner
        to create this particular hdf5 file

dataset name: reg
    type:
        float64
```

(continues on next page)

(continued from previous page)

```
    shape:
        (nvar,)
    description:
        regularization vector. One entry per variable.

dataset name: resolved_tiles
    type (h5py):
        object
    type (HDFView):
        String, length = variable, padding = H5T_STR_NULLTERM,
        cset = H5T_CSET_ASCII
    shape:
        (1,)
    description:
        relative path of resolved tiles (json or json.gz).
        this was easier than trying to embed and encoded dict
        within this file.

dataset name: solve_list
    type:
        int32
    shape:
        (nsolve, 1)
    desctiption:
        overly explicit way to tell the C solver that there are
        1 or two solves. But, it works.

dataset name: x_0
    type:
        float64
    shape:
        (nvar,)
    description:
        input variables for first solve,
        constraint vector for regularizations.

dataset name: x_1
    type:
        float64
    shape:
        (nvar,)
    description:
        input variables for second solve (if present),
        constraint vector for regularizations.
```

format of z1_z2.h5 (one of the distributed A files. z1 and z2 are the min and max section number represented by the block, used to create unique file names)

```
dataset name: data
    type:
       float64
    shape:
       (nnz,)
    description:
       the non-zero sub-matrix entries
```

(continues on next page)

(continued from previous page)

```
dataset name: indices
    type:
        int64
    shape:
        (nnz, 1)
    description:
        the globally-indexed column indices for
        the entries

dataset name: indptr
    type:
        int64
    shape:
        (nrow + 1, 1)
    description:
        index ptr for sub-matrix rows.
        See definition of CSR matrix format.

dataset name: rhs_0
    type:
        float64
    shape:
        (nrow,)
    description:
        right hand side for first solve

dataset name: rhs_1
    type:
        float64
    shape:
        (nrow,)
    description:
        right hand side for second solve

dataset name: rhs_list
    type:
        int32
    shape:
        (nsolve, 1)
    description:
        overly explicit callout for number of solves.

dataset name: weights
    type:
        float64
    shape:
        (nrow,)
    description:
        weight sub-vector
```

Documentation

https://em-aligner-python.readthedocs.io/en/latest/index.html

Author

Dan Kapner e-mail: danielk@alleninstitute.org

CHAPTER 2

API

This contains the complete documentation of the api

2.1 EMaligner

2.1.1 EMaligner schema

class EMaligner.schemas.**EMA_Schema**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: argschema.schemas.ArgSchema

The input schema used by the EM_aligner_python solver

This schema is designed to be a schema_type for an ArgSchemaParser object

Table 1: EMA_Schema

key	description	default	field_type	json_type
input_json	file path of input json file	NA	InputFile	str
output_json	file path to output json file	NA	OutputFile	str
log_level	set the logging level of the module	ERROR	LogLevel	str
first_section	first section for matrix assembly	(REQUIRED)	Integer	int
last_section	last section for matrix assembly	(REQUIRED)	Integer	int
n_parallel_jobs	number of parallel jobs that will run for retrieving tilespecs, assembly from point-matches, and import_tilespecs_parallel	4	Integer	int
processing_chunk_size	number of pairs per multiprocessing job. can help parallelizing pymongo calls.	1	Integer	int
solve_type	Solve type options (montage, 3D)	montage	String	str
close_stack	Set output stack to state COMPLETE?	True	Boolean	bool
overwrite_zlayer	delete section before import tilespecs?	True	Boolean	bool
profile_data_load	module will raise exception after timing tilespec read	False	Boolean	bool
transformation	transformation to use for the solve	AffineModel	String	str
fullsize_transform	use fullsize affine transform	False	Boolean	bool
poly_order	order of polynomial transform.	2	Integer	int
output_mode	none: just solve and show logging output hdf5: assemble to hdf5_options.output_dir stack: write to output stack	none	String	str
assemble_from_file	path to an hdf5 file for solving from hdf5 output.mainly for testing purposes. hdf5 output usually to be solved by external solver		String	str
ingest_from_file	path to an hdf5 file output from the external solver.		String	str
render_output	anything besides the default will show all the render stderr/stdout	null	String	str
input_stack	specifies the origin of the tilespecs.	NA	*input_stack*	dict
output_stack	specifies the destination of the tilespecs.	NA	*output_stack*	dict
pointmatch	specifies the origin of the point correspondences	NA	*pointmatch*	dict
hdf5_options	options invoked if output_mode is 'hdf5'	NA	*hdf5_options*	dict
matrix_assembly	options that control which correspondences are included in the matrix equation and their weights	NA	*matrix_assembly*	dict
regularization	options that contol the regularization of different types of variables in the solve	NA	*regularization*	dict
transform_apply	tilespec.tforms[i].tform() for i in transform_apply will be performed on the matches before matrix assembly.	[]	List	int

opts = <marshmallow.schema.SchemaOpts object>

validate_data(*data*)

class EMaligner.schemas.**input_stack**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: EMaligner.schemas.input_db

Table 2: input_stack

key	description	default	field_type	json_type
owner	render or mongo owner		String	str
project	render or mongo project		String	str
name	render or mongo collection name	NA	List	str
host	render host	NA	String	str
port	render port	8080	Integer	int
mongo_host	mongodb host	em-131fs	String	str
mongo_port	mongodb port	27017	Integer	int
mongo_userName	mongo user name		String	str
mongo_authenticationDatabase	mongo database	admin	String	str
mongo_password	mongo pwd		String	str
db_interface	render: read or write via render mongo: read or write via pymongo file: read or write to file	mongo	String	str
client_scripts	see renderapi.render.RenderClient	/allen/aibs/pipeline/image_processing/volume_assembly/render-jars/production/scripts	String	str
memGB	see renderapi.render.RenderClient	5G	String	str
validate_client	see renderapi.render.RenderClient	False	Boolean	bool
input_file	json or json.gz serialization of input	None	InputFile	str
collection_type	'stack' or 'pointmatch'	stack	String	str
use_rest	passed as arg in import_tilespecs_parallel	False	Boolean	bool

opts = <marshmallow.schema.SchemaOpts object>

validate_data(*data*)

class EMaligner.schemas.**output_stack**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: EMaligner.schemas.db_params

Table 3: output_stack

key	description	default	field_type	json_type
owner	render or mongo owner		String	str
project	render or mongo project		String	str
name	render or mongo collection name	NA	List	str
host	render host	NA	String	str
port	render port	8080	Integer	int
mongo_host	mongodb host	em-131fs	String	str
mongo_port	mongodb port	27017	Integer	int
mongo_userName	mongo user name		String	str
mongo_authenticationDatabase	mongo database	admin	String	str
mongo_password	mongo pwd		String	str
db_interface	render: read or write via render mongo: read or write via pymongo file: read or write to file	mongo	String	str
client_scripts	see renderapi.render.RenderClient	/allen/aibs/pipeline/image_processing/volume_assembly/render-jars/production/scripts	String	str
memGB	see renderapi.render.RenderClient	5G	String	str
validate_client	see renderapi.render.RenderClient	False	Boolean	bool
output_file	json or json.gz serialization of input stackResolvedTiles.	None	OutputFile	str
compress_output	if writing file, compress with gzip.	True	Boolean	bool
collection_type	'stack' or 'pointmatch'	stack	String	str
use_rest	passed as kwarg to renderapi.client.import_tilespecs_parallel	False	Boolean	bool

opts = <marshmallow.schema.SchemaOpts object>

validate_data(*data*)

validate_file(*data*)

class EMaligner.schemas.**pointmatch**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: EMaligner.schemas.input_db

Table 4: pointmatch

key	description	default	field_type	json_type
owner	render or mongo owner		String	str
project	render or mongo project		String	str
name	render or mongo collection name	NA	List	str
host	render host	NA	String	str
port	render port	8080	Integer	int
mongo_host	mongodb host	em-131fs	String	str
mongo_port	mongodb port	27017	Integer	int
mongo_userName	mongo user name		String	str
mongo_authenticationDatabase	mongo database db		String	str
mongo_password	mongo pwd		String	str
db_interface	render: read or write via render mongo: read or write via pymongo file: read or write to file	mongo	String	str
client_scripts	see renderapi.render.RenderClient	/allen/aibs/pipeline/image_processing/volume_assembly/render-jars/production/scripts	String	str
memGB	see renderapi.render.RenderClient	5G	String	str
validate_client	see renderapi.render.RenderClient	False	Boolean	bool
input_file	json or json.gz serialization of input	None	InputFile	str
collection_type	'stack' or 'pointmatch'	pointmatch	String	str

opts = <marshmallow.schema.SchemaOpts object>

class EMaligner.schemas.**hdf5_options**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: argschema.schemas.DefaultSchema

Table 5: hdf5_options

key	description	default	field_type	json_type
output_dir	path to directory to hold hdf5 output.		String	str
chunks_per_file	how many sections with upward-looking cross section to write per .h5 file	5	Integer	int

opts = <marshmallow.schema.SchemaOpts object>

class EMaligner.schemas.**matrix_assembly**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: argschema.schemas.DefaultSchema

Table 6: matrix_assembly

key	description	default	field_type	json_type
depth	depth in z for matrix assembly point matches	[0, 1, 2]	List	int
explicit_weight_by_depth	explicitly set solver weights by depth	None	List	float
cross_pt_weight	weight of cross section point matches	1.0	Float	float
montage_pt_weight	weight of montage point matches	1.0	Float	float
npts_min	disregard any tile pairs with fewer points than this	5	Integer	int
npts_max	truncate any tile pairs to this size	500	Integer	int
choose_random	choose random pts to meet npts_max vs. just first npts_max	False	Boolean	bool
inverse_dz	cross section point match weighting fades with z	True	Boolean	bool

check_explicit(*data*)

opts = <marshmallow.schema.SchemaOpts object>

tolist(*data*)

class EMaligner.schemas.**regularization**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: argschema.schemas.DefaultSchema

Table 7: regularization

key	description	default	field_type	json_type
default_lambda	common regularization value	0.005	Float	float
translation_factor	translation regularization factor. multiplies default_lambda	0.005	Float	float
poly_factors	List of regularization factors by order (0, 1, ..., n) will override other settings for Polynomial2DTransform. multiplies default_lambda	None	List	float
thinplate_factor	regularization factor for thin plate spline control points. multiplies default_lambda.	1e-05	Float	float

opts = <marshmallow.schema.SchemaOpts object>

class EMaligner.schemas.**EMA_PlotSchema**(*extra=None*, *only=None*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

Bases: *EMaligner.schemas.EMA_Schema*

This schema is designed to be a schema_type for an ArgSchemaParser object

Table 8: EMA_PlotSchema

key	description	default	field_type	json_type
input_json	file path of input json file	NA	InputFile	str
output_json	file path to output json file	NA	OutputFile	str
log_level	set the logging level of the module	ERROR	LogLevel	str

Continued on next page

Table 8 – continued from previous page

key	description	default	field_type	json_type
first_section	first section for matrix assembly	(REQUIRED)	Integer	int
last_section	last section for matrix assembly	(REQUIRED)	Integer	int
n_parallel_jobs	number of parallel jobs that will run for retrieving tilespecs, assembly from pointmatches, and import_tilespecs_parallel	4	Integer	int
processing_chunk_size	number of pairs per multiprocessing job. can help parallelizing pymongo calls.	1	Integer	int
solve_type	Solve type options (montage, 3D)	montage	String	str
close_stack	Set output stack to state COMPLETE?	True	Boolean	bool
overwrite_zlayer	delete section before import tilespecs?	True	Boolean	bool
profile_data_load	module will raise exception after timing tilespec read	False	Boolean	bool
transformation	transformation to use for the solve	AffineModel	String	str
fullsize_transform	use fullsize affine transform	False	Boolean	bool
poly_order	order of polynomial transform.	2	Integer	int
output_mode	none: just solve and show logging output hdf5: assemble to hdf5_options.output_dir stack: write to output stack	none	String	str
assemble_from_file	path to an hdf5 file for solving from hdf5 output.mainly for testing purposes. hdf5 output usually to be solved by external solver		String	str
ingest_from_file	path to an hdf5 file output from the external solver.		String	str
render_output	anything besides the default will show all the render stderr/stdout	null	String	str
input_stack	specifies the origin of the tilespecs.	NA	*input_stack*	dict
output_stack	specifies the destination of the tilespecs.	NA	*output_stack*	dict
pointmatch	specifies the origin of the point correspondences	NA	*pointmatch*	dict
hdf5_options	options invoked if output_mode is 'hdf5'	NA	*hdf5_options*	dict
matrix_assembly	options that control which correspondences are included in the matrix equation and their weights	NA	*matrix_assembly*	dict
regularization	options that contol the regularization of different types of variables in the solve	NA	*regularization*	dict
transform_apply	tilespec.tforms[i].tform() for i in transform_apply will be performed on the matches before matrix assembly.	[]	List	int
z1	first z for plot	1000	Integer	int
z2	second z for plot	1000	Integer	int
zoff	z offset between pointmatches and tilespecs	0	Integer	int
plot	make a plot, otherwise, just text output	True	Boolean	bool
savefig	save to a pdf	False	Boolean	bool
plot_dir	no description	./	String	str
threshold	threshold for colors in residual plot [pixels]	5.0	Float	float

Continued on next page

Table 8 – continued from previous page

key	description	default	field_type	json_type
density	whether residual plot is density (for large numbers of points) or just points	True	Boolean	bool

opts = <marshmallow.schema.SchemaOpts object>

2.1.2 EMaligner

class EMaligner.EMaligner.**EMaligner**(*input_data=None*, *schema_type=None*, *output_schema_type=None*, *args=None*, *logger_name='argschema.argschema_parser'*)

Bases: argschema.argschema_parser.ArgSchemaParser

Note: This class takes a ArgSchema as an input to parse inputs , with a default schema of type *EMA_Schema*

assemble_and_solve(*zvals*)
retrieves a ResolvedTiles object from some source and then assembles/solves, outputs to hdf5 and/or outputs to an output_stack object.

Parameters **zvals** (numpy.ndarray) – int or float, z of renderapi.tilespec.TileSpec

assemble_from_db(*zvals*)
assembles a matrix from a pointmatch source given the already-retrieved ResolvedTiles object. Then solves or outputs to hdf5.

Parameters **zvals** – int or float, z of renderapi.tilespec.TileSpec

assemble_from_hdf5(*filename*, *zvals*, *read_data=True*)
assembles and solves from an hdf5 matrix assembly previously created with output_mode = "hdf5".

Parameters **zvals** (numpy.ndarray) – int or float, z of renderapi.tilespec.TileSpec

create_CSR_A(*resolved*)

distributes the work of reading pointmatches and assembling results

Parameters **resolved** (renderapi.resolvedtiles.ResolvedTiles) – resolved tiles object from which to create A matrix

default_schema
alias of *EMaligner.schemas.EMA_Schema*

run()
main function call for EM_aligner_python solver

solve_or_not(*A*, *weights*, *reg*, *x0*, *rhs*)
solves or outputs assembly to hdf5 files

Parameters

- **A** (scipy.sparse.csr) – the matrix, N (equations) x M (degrees of freedom)
- **weights** (scipy.sparse.csr_matrix) – N x N diagonal matrix containing weights

- **reg** (scipy.sparse.csr_matrix) – M x M diagonal matrix containing regularizations
- **x0** (numpy.ndarray) – M x nsolve float constraint values for the DOFs
- **rhs** (numpy.ndarray:) – rhs vector(s) N x nsolve float right-hand-side(s)

Returns

- **message** (*str*) – solver or hdf5 output message for logging
- **results** (*dict*) – keys are "x" (the results), "precision", "error" "err", "mag", and "time"

EMaligner.EMaligner.**calculate_processing_chunk**(*fargs*)
job to parallelize for creating a sparse matrix block and associated vectors from a pair of sections

Parameters **fargs** (*List*) – serialized inputs for multiprocessing job

Returns **chunk** – keys are 'zlist', 'block', 'weights', and 'rhs'

Return type dict

EMaligner.EMaligner.**tilepair_weight**(*z1*, *z2*, *matrix_assembly*)
get weight factor between two tilepairs

Parameters

- **z1** (*int or float*) – z value for first section
- **z2** (*int or float*) – z value for second section
- **matrix_assembly** (*dict*) – EMaligner.schemas.matrix assembly

Returns **tp_weight** – weight factor

Return type float

2.1.3 utils

exception EMaligner.utils.**EMalignerException**
Bases: Exception

EM_aligner exception

EMaligner.utils.**blocks_from_tilespec_pair**(*ptspec*, *qtspec*, *match*, *pcol*, *qcol*, *ncol*, *matrix_assembly*)
create sparse matrix block from tilespecs and pointmatch

Parameters

- **ptspec** (renderapi.tilespec.TileSpec) – ptspec.tforms[-1] is an AlignerTransform object
- **qtspec** (renderapi.tilespec.TileSpec) – qtspec.tforms[-1] is an AlignerTransform object
- **match** (*dict*) – pointmatch between tilepairs
- **pcol** (*int*) – index for start of column entries for p
- **qcol** (*int*) – index for start of column entries for q
- **ncol** (*int*) – total number of columns in sparse matrix
- **matrix_assembly** (*dict*) – see class matrix_assembly in schemas, sets npts

Returns

- **pblock** (scipy.sparse.csr_matrix) – block for the p tilespec/match entry. The full block can be had from pblock - qblock, but, it is a little faster to do vstack and then subtract, so p and q remain separate
- **qblock** (scipy.sparse.csr_matrix) – block for the q tilespec/match entry
- **w** (numpy.ndarray) – weights for the rows in pblock and qblock

EMaligner.utils.**concatenate_results**(*results*)
row concatenates sparse matrix blocks and associated vectors

Parameters **results** (*list*) – dict with keys "block", "weights", "rhs", "zlist"

Returns

- **A** (scipy.sparse.csr_matrix) – the concatenated matrix, N x M
- **weights** (scipy.sparse.csr_matrix) – diagonal matrix containing concatenated weights N x N
- **rhs** (numpy.ndarray) – concatenated rhs vector(s) float. N x nsolve
- **zlist** (numpy.ndarray) – float concatenated z list

EMaligner.utils.**create_or_set_loading**(*stack*)
creates a new stack or sets existing stack to state LOADING

Parameters **stack** (*EMaligner.schemas.output_stack*) –

EMaligner.utils.**determine_zvalue_pairs**(*resolved*, *depths*)
creates a lidt of pairs by z that will be included in the solve

Parameters

- **resolved** (renderapi.resolvedtiles.ResolvedTiles) – input tilespecs
- **depths** (*List*) – depths (z-differences) that will be included in the matrix

Returns **pairs** – keys are z values and sectionIds for each pair

Return type List of dict

EMaligner.utils.**get_matches**(*iId*, *jId*, *collection*, *dbconnection*)
retrieve point correspondences

Parameters

- **iId** (*str*) – sectionId for 1st section
- **jId** (*str*) – sectionId for 2nd section
- **collection** (*EMaligner.schemas.pointmatch*) –
- **dbconnection** (object returned by *EMaligner.utils.make_dbconnection()*) –

Returns **matches** – standard render/mongo representation of point matches

Return type List of dict

EMaligner.utils.**get_resolved_from_z**(*stack*, *tform_name*, *fullsize*, *order*, *z*)

retrieves a ResolvedTiles object from some source and mutates the final transform for each tilespec into an AlignerTransform object

Parameters

- **stack** (*EMaligner.schemas.input_stack*) –

- **tform_name** (*str*) – specifies which transform to mutate into (solve for)
- **fullsize** (*bool*) – passed as kwarg to the EMaligner.transform.AlignerTransform
- **order** (*int*) – passed as kwarg to the EMaligner.transform.AlignerTransform
- **z** (*int or float*) – z value for one section

Returns resolved

Return type renderapi.resolvedtiles.ResolvedTiles

EMaligner.utils.**get_resolved_tilespecs**(*stack*, *tform_name*, *pool_size*, *zvals*, *fullsize=False*, *order=2*)

retrieves ResolvedTiles objects from some source and mutates the final transform for each tilespec into an AlignerTransform object

Parameters

- **stack** (*EMaligner.schemas.input_stack*) –
- **tform_name** (*str*) – specifies which transform to mutate into (solve for)
- **pool_size** (*int*) – level of parallelization for parallel reads
- **fullsize** (*bool*) – passed as kwarg to the EMaligner.transform.AlignerTransform
- **order** (*int*) – passed as kwarg to the EMaligner.transform.AlignerTransform
- **zvals** (numpy.ndarray) – z values for desired sections

Returns resolved

Return type renderapi.resolvedtiles.ResolvedTiles

EMaligner.utils.**get_stderr_stdout**(*outarg*)

helper function for suppressing render output

Parameters **outarg** (*str*) – from input schema "render_output"

Returns **stdeo** – destination for stderr and stdout

Return type file handle or None

EMaligner.utils.**get_z_values_for_stack**(*stack*, *zvals*)

multi-interface wrapper to find overlapping z values between a stack and the requested range.

Parameters

- **stack** (EMaligner.schema.input_stack) –
- **zvals** (numpy.ndarray) – int or float. input z values

Returns **zvals** – int or float. overlapping z values

Return type numpy.ndarray

EMaligner.utils.**make_dbconnection**(*collection*, *which='tile'*, *interface=None*)

creates a multi-interface object for stacks and collections

Parameters

- **collection** (EMaligner.schemas.db_params) –
- **which** (*str*) – switch for having mongo retrieve reference transforms

- **interface** (*str or None*) – specification to override EMaligner.schemas.db_params.db_interface

Returns **dbconnection** – a multi-interface object used by other functions in *EMaligner.utils*

Return type obj

EMaligner.utils.**message_from_solve_results**(*results*)

create summarizing string message about solve for logging

Parameters **results** (*dict*) – returned from *EMaligner.utils.solve()* or read from external solver results

Returns **message** – human-readable summary message

Return type str

EMaligner.utils.**ready_transforms**(*tilespecs*, *tform_name*, *fullsize*, *order*)
mutate last transform in each tilespec to be an AlignerTransform

Parameters

- **tilespecs** (*List*) – renderapi.tilespec.TileSpec objects.
- **tform_name** (*str*) – intended destination type for the mutation
- **fullsize** (*bool*) – passed as kwarg to AlignerTransform
- **order** (*int*) – passed as kwarg to AlignerTransform

EMaligner.utils.**set_complete**(*stack*)
set stack state to COMPLETE

Parameters **stack** (*EMaligner.schemas.output_stack*) –

EMaligner.utils.**solve**(*A*, *weights*, *reg*, *x0*, *rhs*)
regularized, weighted linear least squares solve

Parameters

- **A** (scipy.sparse.csr_matrix) – the matrix, N (equations) x M (degrees of freedom)
- **weights** (scipy.sparse.csr_matrix) – N x N diagonal matrix containing weights
- **reg** (scipy.sparse.csr_matrix) – M x M diagonal matrix containing regularizations
- **x0** (numpy.ndarray) – M x nsolve float constraint values for the DOFs
- **rhs** (numpy.ndarray) – rhs vector(s) N x nsolve float right-hand-side(s)

Returns **results** – includes solution "x" and summary metrics

Return type dict

EMaligner.utils.**transform_match**(*match*, *ptspec*, *qtspec*, *apply_list*, *tforms*)

transform the match coordinates through a subset of the tilespec transform list

Parameters

- **match** (*dict*) – one match object
- **ptspec** (renderapi.tilespec.TileSpec) – the tilespec for the p coordinates
- **qtspec** (renderapi.tilespec.TileSpec) – the tilespec for the q coordinates

- **apply_list** (*list*) – list of indices for the transforms
- **tforms** (*list*) – list of reference transforms

Returns **match** – one match object, with p and q transformed

Return type dict

EMaligner.utils.**update_tilespecs**(*resolved*, *x*)
update tilespecs with new solution

Parameters

- **resolved** (renderapi.resolvedtiles.ResolvedTiles) – resolved tilespecs to update
- **x** (numpy.ndarray) – results of solve

EMaligner.utils.**write_chunk_to_file**(*fname*, *c*, *file_weights*, *rhs*)
write a sub-matrix to an hdf5 file for an external solve

Parameters

- **fname** (*str*) – path to output file
- **c** (scipy.sparse.csr_matrix) – N x M matrix sub block
- **file_weights** (numpy.ndarray) – length N array of weights
- **rhs** (numpy.ndarray) – N x nsolve right hand sides

EMaligner.utils.**write_reg_and_tforms**(*args*, *resolved*, *metadata*, *tforms*, *reg*)
write regularization and transforms (x0) to hdf5

Parameters

- **args** (*dict*) – passed from EMaligner object
- **resolved** (renderapi.resolvedtiles.ResolvedTiles) – resolved tilespec object to output
- **metadata** (*dict*) – helper values about matrix for external solver
- **tforms** (numpy.ndarray) – M x nsolve starting values (x0)
- **reg** (scipy.sparse.csr_matrix) – M x M diagonal regularization values

EMaligner.utils.**write_to_new_stack**(*resolved*, *output_stack*, *outarg*, *overwrite_zlayer*, *args*, *results*)
write results to render or file output

Parameters

- **resolved** (renderapi.resolvedtiles.ResolvedTiles) – resolved tilespecs containing tilespecs to write
- **output_stack** (*dict*) – from *EMaligner.schemas.output_stack*
- **outarg** (*str*) – render_output argument
- **overwrite_zlayer** (*bool*) – delete section first before overwriting?
- **args** (*dict*) – from *EMaligner.schemas.EMA_Schema*
- **results** (*dict*) – results from *EMaligner.utils.solve()*

Returns **output_stack** – representation of *EMaligner.schemas.output_stack*

Return type dict

2.1.4 transforms

The EMaligner solver uses AlignerTransform objects to construct linear least squares elements from tilespecs. The final transform in each tilespec transform list is converted into an AlignerTransform object, which has a renderapi.transform.transform object as its base class. The AlignerTransform object has a few other methods to enable constructing the matrix, setting regularizations, extracting the starting transform values and setting the solved transform values.

class EMaligner.transform.transform.**AlignerTransform**(*name=None*, *transform=None*, *fullsize=False*, *order=2*)

Bases: object

general transform object that the solver expects

__init__(*name=None*, *transform=None*, *fullsize=False*, *order=2*)

Parameters

- **name** (*str*) – specifies the intended transform for the type of solve
- **transform** (renderapi.transform.Transform) – The new AlignerTransform will inherit from this transform, if possible.
- **fullsize** (*bool*) – only applies to affine transform. Remains for legacy reason as an explicit demonstration of the equivalence of fullsize and halfsize transforms.
- **order** (*int*) – used in Polynomial2DTransform

class EMaligner.transform.affine_model.**AlignerAffineModel**(*transform=None*, *fullsize=False*)

Bases: renderapi.transform.leaf.affine_models.AffineModel

Object for implementing full or half-size affine transforms

__init__(*transform=None*, *fullsize=False*)

Parameters

- **transform** (renderapi.transform.Transform) – The new AlignerTransform will inherit from this transform, if possible.
- **fullsize** (*bool*) – only applies to affine transform. Remains for legacy reason as an explicit demonstration of the equivalence of fullsize and halfsize transforms.

block_from_pts(*pts*, *w*, *col_ind*, *col_max*)

partial sparse block for a transform/match

Parameters

- **pts** (numpy.ndarray) – N x 2, the x, y values of the match (either p or q)
- **w** (numpy.ndarray) – size N, the weights associated with the pts
- **col_ind** (*int*) – the starting column index for this tile
- **col_max** (*int*) – total number of columns in the matrix

Returns

- **block** (scipy.sparse.csr_matrix) – the partial block for this transform
- **w** (numpy.ndarray) – the weights associated with the rows of this block
- **rhs** (numpy.ndarray) – N/2 x 2 (halfsize) or N x 1 (fullsize) right hand side for this transform. generally all zeros. could implement fixed tiles in rhs later.

from_solve_vec(*vec*)
reads values from solution and sets transform parameters

Parameters **vec** (numpy.ndarray) – input to this function is sliced so that vec[0] is the first harvested value for this transform

Returns **n** – number of rows read from vec. Used to increment vec slice for next transform

Return type int

regularization(*regdict*)
regularization vector from this transform

Parameters **regdict** (*dict*) – EMaligner.schemas.regularization. controls regularization values

Returns **reg** – array of regularization values of length DOF_per_tile

Return type numpy.ndarray

to_solve_vec()
sets solve vector values from transform parameters

Returns **vec** – N/2 x 2 for halfsize, N x 2 for fullsize

Return type numpy.ndarray

class EMaligner.transform.polynomial_model.**AlignerPolynomial2DTransform**(*transform=None*, *order=2*)
Bases: renderapi.transform.leaf.polynomial_models.Polynomial2DTransform

Object for implementing half-size polynomial transforms

__init__(*transform=None*, *order=2*)

Parameters

- **transform** (renderapi.transform.Transform) – The new AlignerTransform will inherit from this transform, if possible.
- **order** (*int*) – order of the intended polynomial.

block_from_pts(*pts*, *w*, *col_ind*, *col_max*)
partial sparse block for a transform/match

Parameters

- **pts** (numpy.ndarray) – N x 2, the x, y values of the match (either p or q)
- **w** (numpy.ndarray) – the weights associated with the pts
- **col_ind** (*int*) – the starting column index for this tile
- **col_max** (*int*) – number of columns in the matrix

Returns

- **block** (scipy.sparse.csr_matrix) – the partial block for this transform
- **w** (numpy.ndarray) – the weights associated with the rows of this block
- **rhs** (numpy.ndarray) – N/2 x 2 (halfsize) or N x 1 (fullsize) right hand side for this transform. generally all zeros. could implement fixed tiles in rhs later.

from_solve_vec(*vec*)
reads values from solution and sets transform parameters

Parameters **vec** (numpy.ndarray) – input to this function is sliced so that vec[0] is the first harvested value for this transform

Returns **n** – number of rows read from vec. Used to increment vec slice for next transform

Return type int

regularization(*regdict*)
regularization vector

Parameters **regdict** (*dict*) – EMaligner.schemas.regularization. controls regularization values

Returns **reg** – array of regularization values of length DOF_per_tile

Return type numpy.ndarray

to_solve_vec()
sets solve vector values from transform parameters

Returns **vec** – N x 2 transform parameters in solve form

Return type numpy.ndarray

class EMaligner.transform.rotation_model.**AlignerRotationModel**(*transform=None*)
Bases: renderapi.transform.leaf.affine_models.AffineModel

Object for implementing rotation transform

__init__(*transform=None*)

Parameters **transform** (renderapi.transform.Transform) – The new AlignerTransform will inherit from this transform, if possible.

block_from_pts(*pts*, *w*, *col_ind*, *col_max*)

partial sparse block for a transform/match. Note: for rotation, a pre-processing step is called at the tilepair level.

Parameters

- **pts** (numpy.ndarray) – N x 1, preprocessed from preprocess()
- **w** (numpy.ndarray) – the weights associated with the pts
- **col_ind** (*int*) – the starting column index for this tile
- **col_max** (*int*) – number of columns in the matrix

Returns

- **block** (scipy.sparse.csr_matrix) – the partial block for this transform
- **w** (numpy.ndarray) – the weights associated with the rows of this block
- **rhs** (numpy.ndarray) – N x 1 right hand side for this transform.

from_solve_vec(*vec*)
reads values from solution and sets transform parameters

Parameters **vec** (numpy.ndarray) – input to this function is sliced so that vec[0] is the first harvested value for this transform

Returns **n** – number of rows read from vec. Used to increment vec slice for next transform

Return type int

static preprocess(*ppts*, *qpts*, *w*)

tilepair-level preprocessing step for rotation transform. derives the relative center-of-mass angles between all p's and q's to avoid angular discontinuity. Will filter out points very close to center-of-mass. Tilepairs with relative rotations near 180deg will not avoid the discontinuity.

Parameters

- **ppts** (numpy.ndarray) – N x 2. The p tile correspondence coordinates
- **qpts** (numpy.ndarray) – N x 2. The q tile correspondence coordinates
- **w** (numpy.ndarray) – size N. The weights.

Returns

- **pa** (numpy.ndarray) – M x 1 preprocessed angular distances. -0.5 x delta angle M <= N depending on filter
- **qa** (numpy.ndarray) – M x 1 preprocessed angular distances. 0.5 x delta angle M <= N depending on filter
- **w** (numpy.ndarray) – size M. filtered weights.

regularization(*regdict*)
regularization vector

Parameters **regdict** (*dict*) – EMaligner.schemas.regularization. controls regularization values

Returns **reg** – array of regularization values of length DOF_per_tile

Return type numpy.ndarray

to_solve_vec()
sets solve vector values from transform parameters

Returns **vec** – N x 1 transform parameters in solve form

Return type numpy.ndarray

class EMaligner.transform.similarity_model.**AlignerSimilarityModel**(*transform=None*)
Bases: renderapi.transform.leaf.affine_models.AffineModel

Object for implementing similarity transform.

__init__(*transform=None*)

Parameters **transform** (renderapi.transform.Transform) – The new AlignerTransform will inherit from this transform, if possible.

block_from_pts(*pts*, *w*, *col_ind*, *col_max*)

partial sparse block for a transform/match. similarity constrains the center-of-mass coordinates to transform according to the same affine transform as the coordinates, save translation.

Parameters

- **pts** (numpy.ndarray) – N x 2, the x, y values of the match (either p or q)
- **w** (numpy.ndarray) – the weights associated with the pts
- **col_ind** (*int*) – the starting column index for this tile
- **col_max** (*int*) – number of columns in the matrix

Returns

- **block** (scipy.sparse.csr_matrix) – the partial block for this transform
- **w** (numpy.ndarray) – the weights associated with the rows of this block
- **rhs** (numpy.ndarray) – N x 1 (fullsize) right hand side for this transform. generally all zeros. could implement fixed tiles in rhs later.

from_solve_vec(*vec*)
reads values from solution and sets transform parameters

Parameters **vec** (numpy.ndarray) – input to this function is sliced so that vec[0] is the first harvested value for this transform

Returns **n** – number of rows read from vec. Used to increment vec slice for next transform

Return type int

regularization(*regdict*)
regularization vector

Parameters **regdict** (*dict*) – EMaligner.schemas.regularization. controls regularization values

Returns **reg** – array of regularization values of length DOF_per_tile

Return type numpy.ndarray

to_solve_vec()
sets solve vector values from transform parameters

Returns **vec** – N x 1 transform parameters in solve form

Return type numpy.ndarray

class EMaligner.transform.thinplatespline_model.**AlignerThinPlateSplineTransform**(*transform=Non*
Bases: renderapi.transform.leaf.thin_plate_spline.ThinPlateSplineTransform

Object for implementing thin plate spline transform

__init__(*transform=None*)

Parameters **transform** (renderapi.transform.Transform) – The new Aligner-Transform will inherit from this transform, if possible.

block_from_pts(*pts*, *w*, *col_ind*, *col_max*)
partial sparse block for a tilepair/match

Parameters

- **pts** (numpy.ndarray) – N x 2, the x, y values of the match (either p or q)
- **w** (numpy.ndarray) – the weights associated with the pts
- **col_ind** (*int*) – the starting column index for this tile
- **col_max** (*int*) – number of columns in the matrix

Returns

- **block** (scipy.sparse.csr_matrix) – the partial block for this transform
- **w** (numpy.ndarray) – the weights associated with the rows of this block
- **rhs** (numpy.ndarray) – N/2 x 2 right hand side for this transform.

from_solve_vec(*vec*)
reads values from solution and sets transform parameters

Parameters **vec** (numpy.ndarray) – input to this function is sliced so that vec[0] is the first harvested value for this transform

Returns **n** – number of rows read from vec. Used to increment vec slice for next transform

Return type int

regularization(*regdict*)
regularization vector

Parameters **regdict** (*dict*) – EMaligner.schemas.regularization. controls regularization values

Returns **reg** – array of regularization values of length DOF_per_tile

Return type numpy.ndarray

scale
tuple of scale for x, y. For setting regularization, it is useful to watch scale (logged output for the solver) to look for unwanted distortions and shrinking. Other transforms have scale implemented inside of renderapi.

to_solve_vec()
sets solve vector values from transform parameters

Returns **vec** – N x 2 transform parameters in solve form

Return type numpy.ndarray

class EMaligner.transform.translation_model.**AlignerTranslationModel**(*transform=None*)
Bases: renderapi.transform.leaf.affine_models.AffineModel

Object for implementing translation transform

__init__(*transform=None*)

Parameters **transform** (renderapi.transform.Transform) – The new AlignerTransform will inherit from this transform, if possible.

block_from_pts(*pts*, *w*, *col_ind*, *col_max*)
partial sparse block for a tilepair/match

Parameters

- **pts** (numpy.ndarray) – N x 2, the x, y values of the match (either p or q)
- **w** (numpy.ndarray) – the weights associated with the pts
- **col_ind** (*int*) – the starting column index for this tile
- **col_max** (*int*) – number of columns in the matrix

Returns

- **block** (scipy.sparse.csr_matrix) – the partial block for this transform
- **w** (numpy.ndarray) – the weights associated with the rows of this block
- **rhs** (numpy.ndarray) – N/2 x 2 right hand side for this transform.

from_solve_vec(*vec*)
reads values from solution and sets transform parameters

Parameters **vec** (numpy.ndarray) – input to this function is sliced so that vec[0] is the first harvested value for this transform

Returns **n** – number of rows read from vec. Used to increment vec slice for next transform

Return type int

regularization(*regdict*)
regularization vector

Parameters **regdict** (*dict*) – EMaligner.schemas.regularization. controls regularization values

Returns **reg** – array of regularization values of length DOF_per_tile

Return type numpy.ndarray

to_solve_vec()
sets solve vector values from transform parameters

Returns **vec** – 1 x 2 transform parameters in solve form

Return type numpy.ndarray

exception EMaligner.transform.utils.**AlignerTransformException**
Bases: Exception

Exception class for AlignerTransforms

EMaligner.transform.utils.**aff_matrix**(*theta*, *offs=None*)
affine matrix or augmented affine matrix given a rotation angle.

Parameters

- **theta** (*float*) – rotation angle in radians
- **offs** (numpy.ndarray) – the translations to include

Returns **M** – 2 x 2 (for offs=None) affine matrix or 3 x 3 augmented matrix

Return type numpy.ndarray

2.1.5 jsongz

EMaligner.jsongz.**dump**(*obj*, *filepath*, *compress=None*, *encoding='utf-8'*, **args*, ***kwargs*)
json or json.gz dump

Parameters

- **obj** (*obj*) – object to dump
- **filepath** (*str*) – path for destination of dump
- **compress** (*bool or None*) – if None, file compressed or not according to filepath extension
- **encoding** (*str*) – encoding of json.dumps() before writing to .gz file. not passed into json.dump()
- ***args** – json.dump() args
- ****kwargs** – json.dump() kwargs

Returns **filepath** – potentially modified filepath of dumped object uncompressed are forced to '.json' and compressed to '.gz'

Return type str

EMaligner.jsongz.**load**(*filepath*, *encoding='utf-8'*, **args*, ***kwargs*)
json or json.gz load

Parameters

- **filepath** (*str*) – path for source of load
- **encoding** (*str*) – encoding for decoding of `json.dumps()` after .gz read not passed into `json.load()`
- ***args** – `json.load()` args
- ****kwargs** – `json.load()` kwargs

Returns **obj** – loaded object

Return type dict

2.2 EM aligner distributed

2.2.1 Class Hierarchy

2.2.2 File Hierarchy

2.2.3 Full API

Classes and Structs

Struct node

- Defined in file_EMaligner_distributed_src_hw_config.h

Struct Documentation

struct node

Public Members

char ***name**

double **mem**

int ***ranks**

int **nrank**

double **tmem**

int **nfiles**

int ***files**

Functions

Function CopyDataSetstoSolutionOut

- Defined in file_EMaligner_distributed_src_ema.h

Function Documentation

void **CopyDataSetstoSolutionOut**(MPI_Comm *COMM*, char *indexname*[], char *outputname*[])
output file of a solve is mostly a copy of the input file

Parameters

- `COMM`: The MPI communicator, probably PETSC_COMM_WORLD.
- `indexname`: The full path to <solution_input>.h5, given as command-line argument with -input.
- `outputname`: The full path to <solution_output>.h5, given as command-line argument with -output.

Function CountFiles

- Defined in file_EMaligner_distributed_src_ema.h

Function Documentation

PetscErrorCode **CountFiles**(MPI_Comm *COMM*, char *indexname*[], int **nfiles*)
Rank 0 process counts the number of files from index.txt and broadcasts the result

Parameters

- `COMM`: The MPI communicator, probably PETSC_COMM_WORLD.
- `indexname`: The full path to index.txt, given as command-line argument with -f.
- `*nfiles`: The result of the count, returned to main().

Function CountSolves

- Defined in file_EMaligner_distributed_src_ema.h

Function Documentation

PetscErrorCode **CountSolves**(MPI_Comm *COMM*, char *indexname*[], PetscInt **nsolve*)
Counts how many x0 vectors are stored in the regularization file.

Parameters

- `COMM`: The MPI communicator, PETSC_COMM_SELF.A
- `dir`: string containing the directory
- `number`: of solves

Function CreateL

- Defined in file_EMaligner_distributed_src_ema.h

Function Documentation

PetscErrorCode **CreateL**(MPI_Comm *COMM*, char *indexname*[], PetscInt *local_nrow*, PetscInt *global_nrow*, PetscBool *trunc*, Mat **L*)

Creates a diagonal matrix with the weights as the entries.

Parameters

- `COMM`: The MPI communicator, PETSC_COMM_SELF.A
- `local_nrow`: Number of rows for this rank
- `local_row0`: The starting row for this rank.
- `global_nrow`: The total length of the weights vector (N). L is NxN.
- `*L`: The matrix created by this function.

Function CreateW

- Defined in file_EMaligner_distributed_src_ema.h

Function Documentation

PetscErrorCode **CreateW**(MPI_Comm *COMM*, PetscScalar **local_weights*, PetscInt *local_nrow*, PetscInt *local_row0*, PetscInt *global_nrow*, Mat **W*)

Creates a diagonal matrix with the weights as the entries.

Parameters

- `COMM`: The MPI communicator, PETSC_COMM_SELF.A
- `*local_weights`: Passed into this function to get built into W.
- `local_nrow`: Number of rows for this rank
- `local_row0`: The starting row for this rank.
- `global_nrow`: The total length of the weights vector (N). W is NxN.
- `*W`: The matrix created by this function.

Function disp_node

- Defined in file_EMaligner_distributed_src_hw_config.h

Function Documentation

void **disp_node**(*node inode*)

Function freenode

- Defined in file_EMaligner_distributed_src_hw_config.h

Function Documentation

void **freenode**(*node inode*)

Function GetGlobalLocalCounts

- Defined in file_EMaligner_distributed_src_ema.h

Function Documentation

void **GetGlobalLocalCounts**(int *nfiles*, PetscInt ***metadata*, int *local_firstfile*, int *local_lastfile*, PetscInt **global_nrow*, PetscInt **global_ncol*, PetscInt **global_nnz*, PetscInt **local_nrow*, PetscInt **local_nnz*, PetscInt **local_row0*)

Use metadata to determine global and local sizes and indices.

Parameters

- `nfiles`: The number of CSR.hdf5 files, from a previous call to *CountFiles()*
- `**metadata`: Metadata from index.txt from a previous call to ReadIndex()
- `local_firstfile`: first index in the list of files for this rank
- `local_lastfile`: last index in the list of files for this rank
- `global_nrow`: total number of rows from metadata
- `global_ncol`: total number of columns from metadata
- `global_nnz`: total number of non-zero entries from metadata
- `local_nrow`: number of rows for this rank
- `local_nnz`: number of non-zero entries for this rank
- `local_row0`: index of first row for this rank

Function hw_config

- Defined in file_EMaligner_distributed_src_hw_config.h

Function Documentation

node ***hw_config**(MPI_Comm *COMM*, int **nnodes*, int **thisnode*)

Function index_rank

- Defined in file_EMaligner_distributed_src_hw_config.h

Function Documentation

node ***index_rank** (char **names*, double **mem*, int *n*, char **unames*, int *un*, int *mx*)

Function main

- Defined in file_EMaligner_distributed_src_em_dist_solve.c

Function Documentation

Warning: doxygenfunction: Cannot find function "main" in doxygen xml output for project "ema_distributed" from directory: ./doxyoutput/xml

Function ReadIndexSet

- Defined in file_EMaligner_distributed_src_ema.h

Function Documentation

PetscErrorCode **ReadIndexSet** (MPI_Comm *COMM*, PetscViewer *viewer*, char **varname*, IS **newIS*, PetscInt **n*)

Read data from a PetscViewer into an IS (Index Set, i.e. integer) object.

Parameters

- `COMM`: The MPI communicator, PETSC_COMM_SELF or PETSC_COMM_WORLD.
- `viewer`: A PetscViewer instance.
- `*varname`: The new IS will have this name. For reading from hdf5 files, this name must match the dataset name in the file.
- `*newIS`: The new IS object.
- `*n`: The number of entries in the new object.

Function ReadLocalCSR

- Defined in file_EMaligner_distributed_src_ema.h

Function Documentation

PetscErrorCode **ReadLocalCSR** (MPI_Comm *COMM*, char **csrnames*[], int *local_firstfile*, int *local_lastfile*, int *nsolve*, PetscInt **local_indptr*, PetscInt **local_jcol*, PetscScalar **local_data*, PetscScalar **local_weights*, PetscScalar ***local_rhs*)

Build local CSR block by sequentially reading in local hdf5 files.

Parameters

- `COMM`: The MPI communicator, PETSC_COMM_SELF.

- `*csrnames[]`: The names of the CSR.hdf5 files
- `local_firstfilelocal_lastfile`: Indices of which files are handled by which rank.
- `nsolve`: how many solves (right-hand sides) to perform
- `*local_indptr`: CSR index ptr for this rank
- `*local_jcol`: CSR column indices for this rank
- `*local_data`: CSR data for this rank
- `*local_weights`: Holds the concatenated weights for this rank.
- `*local_rhs`: Holds the right-hand-side(s) for this rank

Function ReadMetadata

- Defined in file_EMaligner_distributed_src_ema.h

Function Documentation

PetscErrorCode **ReadMetadata**(MPI_Comm *COMM*, char *indexname*[], int *nfiles*, char **csrnames*[], PetscInt ***metadata*)

Rank 0 processor reads metadata and broadcasts.

Parameters

- `COMM`: The MPI communicator, probably PETSC_COMM_WORLD.
- `indexname`: The full path to <solution_input>.h5, given as command-line argument with -input.
- `nfiles`: The number of files listed in solution_input.h5 file, read from previous *CountFiles()* call.
- `*csrnames`: An array of file names from <solution_input>.h5, populated by this function.
- `*metadata`: An array of metadata from index.txt, populated by this function.

Function ReadVec

- Defined in file_EMaligner_distributed_src_ema.h

Function Documentation

PetscErrorCode **ReadVec**(MPI_Comm *COMM*, PetscViewer *viewer*, char **varname*, Vec **newvec*, PetscInt **n*)

Read data from a PetscViewer into a Vec (scalar) object. This version good for single-rank reads.

Parameters

- `COMM`: The MPI communicator, PETSC_COMM_SELF or PETSC_COMM_WORLD.
- `viewer`: A PetscViewer instance.
- `*varname`: The new vector will have this name. For reading from hdf5 files, this name must match the dataset name in the file.
- `*newvec`: The new vector object.
- `*n`: The number of entries in the new object.

Function ReadVecWithSizes

- Defined in file_EMaligner_distributed_src_ema.h

Function Documentation

PetscErrorCode **ReadVecWithSizes**(MPI_Comm *COMM*, PetscViewer *viewer*, char **varname*, Vec **newvec*, PetscInt **n*, PetscInt *nlocal*, PetscInt *nglobal*, PetscBool *trunc*)

Read data from a PetscViewer into a Vec (scalar) object. This version good for anticipating allocation across nodes.

Parameters

- `COMM`: The MPI communicator, PETSC_COMM_SELF or PETSC_COMM_WORLD.
- `viewer`: A PetscViewer instance.
- `*varname`: The new vector will have this name. For reading from hdf5 files, this name must match the dataset name in the file.
- `*newvec`: The new vector object.
- `*n`: The number of entries in the new object.
- `nlocal`: The number of entries the local rank will own.
- `nglobal`: The total number of expected entries in newvec.
- `trunc`: Boolean whether to truncate the imported data.

Function Readx0

- Defined in file_EMaligner_distributed_src_ema.h

Function Documentation

PetscErrorCode **Readx0**(MPI_Comm *COMM*, char *indexname*[], PetscInt *local_nrow*, PetscInt *global_nrow*, PetscInt *nsolve*, PetscBool *trunc*, Vec *x0*[])

Read the x0 vectors stored in the regularization file.

Parameters

- `COMM`: The MPI communicator, PETSC_COMM_SELF.A
- `dir`: string containing the directory
- `local_nrow`: Number of local rows this rank will own
- `global_nrow`: Number of global rows.
- `nsolve`: Number of x0 vectors to be read.
- `x0[]`: vectors

Function SetFiles

- Defined in file_EMaligner_distributed_src_ema.h

Function Documentation

PetscErrorCode **SetFiles**(MPI_Comm *COMM*, int *nfiles*, PetscInt **firstfile*, PetscInt **lastfile*)
Split the list of files roughly evenly amongst all the workers.

Parameters

- `COMM`: The MPI communicator, probably PETSC_COMM_WORLD.
- `nfiles`: The number of lines in index.txt, read from previous *CountFiles()* call.
- `*firstfile`: The index of the first file for this worker.
- `*lastfile`: The index of the first file for this worker.

Function ShowMatInfo

- Defined in file_EMaligner_distributed_src_ema.h

Function Documentation

PetscErrorCode **ShowMatInfo**(MPI_Comm *COMM*, Mat **m*, **const** char **mesg*)
Print to stdout MatInfo for a Mat object. Caution: hangs on multi-node. Don't use until fixed.

Parameters

- `COMM`: The MPI communicator PETSC_COMM_WORLD.
- `*m`: The matrix.
- `*mesg`: Name or some other string to prepend the output.

Function split_files

- Defined in file_EMaligner_distributed_src_hw_config.h

Function Documentation

void **split_files**(*node* **nodes*, int *nnodes*, int *nfiles*)

Function unique_str

- Defined in file_EMaligner_distributed_src_hw_config.h

Function Documentation

char ***unique_str**(char **in*, int *n*, int *m*, int **nu*)

Variables

Variable help

- Defined in file_EMaligner_distributed_src_em_dist_solve.c

Variable Documentation

Warning: doxygenvariable: Cannot find variable "help" in doxygen xml output for project "ema_distributed" from directory: ./doxyoutput/xml

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

CHAPTER 4

References

CHAPTER 5

Acknowledgement of Government Sponsorship

Supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior / Interior Business Center (DoI/IBC) contract number D16PC00004. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/IBC, or the U.S. Government.

Bibliography

[Kapner19] Daniel Kapner. . . Tbd. *TBD*, 2019. URL: TBD, arXiv:TBD.

[KDS18] Khaled Khairy, Gennady Denisov, and Stephan Saalfeld. Joint deformable registration of large EM image volumes: A matrix solver approach. *CoRR*, 2018. URL: http://arxiv.org/abs/1804.10019, arXiv:1804.10019.

[Mahalingam19] Gayathri Mahalingam. . . Tbd. *TBD*, 2019. URL: TBD, arXiv:TBD.

Python Module Index

e

Index

Symbols

A

B

C

S

T

U

V

W