BDDCML

solver library based on Multi-Level Balancing Domain Decomposition by Constraints

version 1.2

Jakub Šístek

Table of Contents

1 Introduction

The BDDCML (Balancing Domain Decomposition by Constraints - Multi-Level) is a library for solving large sparse linear systems resulting from computations by the finite element method (FEM). Domain decomposition technique is employed which allows distribution of the computations among processors.

The main goal of the package is to provide a scalable implementation of the (Adaptive) Multilevel BDDC method. Codes are written in Fortran 95 with MPI library. A library is provided, which is supposed to be called from users' applications. It provides a simple interface functions callable from Fortran and C.

Balancing Domain Decomposition by Constraints (BDDC) has quickly evolved into a very popular method. However, for very large numbers of subdomain, the coarse problem becomes a large problem to be solved in its own right. In Multilevel BDDC, the coarse problem is solved only approximately by recursive application of BDDC to higher levels.

The main web site of the BDDCML project is

http://www.math.cas.cz/~sistek/software/bddcml.html

In case of questions, reporting a bug, or just of interest, feel free to contact Jakub Šístek at sistek@math.cas.cz.

2 How to use BDDCML

The library provides a simple interface callable from Fortran and C. Although main parts of the solver are purely algebraic, the solver needs also to get some information of the computational mesh. This requirement is mainly motivated by selection of corners within the method, for which existing algorithms rely on geometry.

Two different modes are possible for input:

- user can either provide information about global mesh and a file with element matrices (global loading),
- user can provide division into subdomains on the first level and pass subdomain matrices for each subdomain to the routine (local loading).

The solution process is divided into the following sequence of called functions. Their parameters are described in separate sections.

1. `bddcml_init` – initialization of the solver
2.
   - `bddcml_upload_global_data` – loading global data about computational mesh and matrix (use for global loading)
   - `bddcml_upload_subdomain_data` – loading data for one subdomain mesh and matrix (use for local loading)
3. `bddcml_setup_preconditioner` – prepare preconditioner
4. `bddcml_solve` – solve loaded system by precodnitioned Krylov subspace iterative method (PCG or BiCGstab)
5.
   - `bddcml_download_local_solution` – get the solution restricted to a subdomain from the solver (use for local loading)
   - `bddcml_download_global_solution` – get the global solution from the solver (use for global loading)
6. `bddcml_finalize` – clear solver data and deallocate memory

Two examples are presented in the `examples` folder. Both are written in Fortran 90. The '`bddcml_global.f90`' demonstrates the use of global input, while the '`bddcml_local.f90`' demonstrates the use of localized subdomain input.

3 Description of interface functions

In this chapter, detailed description of the solver interface functions with explanation of individual arguments is given

3.1 bddcml_init

C interface

`void bddcml_init( int *nl, int *nsublev, int *lnsublev, int *nsub_loc_1, int *comm_init, int *verbose_level, int *numbase )`

Description

Prepares internal data structures for the solver.

Parameters

`nl` given number of levels

`nsublev` array with GLOBAL numbers of subdomains for each level

`lnsublev` length of array `nsublev` - should match `nl`

`nsub_loc_1`
LOCAL number of subdomains assigned to the process.
- >= 0 number of local subdomains - sum up across processes to nsublev[0]
- -1 let solver decide, the value is returned (determining linear partition)

`comm_init`
initial global communicator (possibly `MPI_COMM_WORLD`). This should be communicator in Fortran. When called from C, it should NOT be of type `MPI_Comm`. Use `MPI_Comm_c2f` function before calling this routine to get the proper argument.

`verbose_level`
level of verbosity
- 0 - only errors printed
- 1 - some output
- 2 - detailed output

`numbase` first index of arrays (0 for C, 1 for Fortran)

3.2 bddcml_upload_global_data

C interface

void bddcml_upload_global_data(int *nelem, int *nnod, int *ndof, int *ndim, int *meshdim, int *inet, int *linet, int *nnet, int *lnnet, int *nndf, int *lnndf, double *xyz, int *lxyz1, int *lxyz2, int *ifix, int *lifix, double *fixv, int *lfixv, double *rhs, int *lrhs, double *sol, int *lsol, int *idelm, int *neighbouring, int *load_division_int)

Description

If no distribution of data exists in the user application, it may be left to the solver. This routine loads global information on mesh connectivity and coordinates. Matrix is passed as unassembled matrices of individual elements which will be read from opened file unit `idelm` and assembled within the solver. If partitionining into subdomains on the basic level exists in user's application, routine `bddcml_upload_subdomain_data` should be used instead.

Parameters

`nelem` GLOBAL number of elements

`nnod` GLOBAL number of nodes

`ndof` GLOBAL number of degrees of freedom, i.e. size of matrix

`ndim` number of space dimensions

`meshdim` mesh dimension. For 3D elements = `ndim`, for 3D shells = 2, for 3D beams = 1

`inet` GLOBAL array with Indices of Nodes on ElemenTs - this defines connectivity of the mesh.

`linet` length of array `inet`. It is given as a sum of entries in array `nnet`.

`nnet` GLOBAL array with Number of Nodes on ElemenTs. For each element, it gives number of nodes it is connected to. This is important to locate element entries in array `inet`

`lnnet` length of array `nnet`. It is equal to `nelem`.

`nndf` GLOBAL array with Number of Nodal Degrees of Freedom. For each node, it gives number of attached degrees of freedom.

`lnndf` length of array `nndf`. It is equal to `nnod`.

`xyz` GLOBAL Coordinates of nodes as one array (all X, all Y, all Z) or as two-dimensional array in Fortran (X | Y | Z). Rows are defined by nodes, columns are defined by dimension.

`lxyz1,lxyz2`
dimensions of array `xyz`. In C, lenght of `xyz` is defined as `lxyz1 * lxyz2`. In Fortran, dimension of `xyz` is given used as `xyz(lxyz1,lxyz2)`. The `lxyz1` is equal to `nnod`. The `lxyz2` is equal to `ndim`.

`ifix` GLOBAL array of Indices of FIXed variables - all degrees of freedom with Dirichlet BC are marked with its number, i.e. non-zero entries determine fixed degrees of freedom.

`lifix` length of array `ifix`, equal to `ndof`.

`fixv` GLOBAL array of FIXed Variables - where `ifix` is non-zero, `fixv` stores value of Dirichlet boundary condition. Where `ifix` is zero, corresponding value in `fixv` is meaningless.

`lfixv` length of array `fixv`, equal to `ndof`.

`rhs` GLOBAL array with Right-Hand Side

`lrhs` length of array `rhs`, equal to `ndof`.

`sol` GLOBAL array with initial SOLution guess. This is used as initial approximation for iterative method.

`lsol` length of array `sol`, equal to `ndof`.

`idelm` opened Fortran unit with unformatted file with element matrices

`neighbouring`
how many nodes should be shared by two elements to call them adjacent in graph. This parameter is used for division of mesh on the basic level by `ParMETIS` or `METIS`. Often, one gets better results if he specifies this number to define adjacency only if elements share a face in 3D or edge in 2D. E.g. for linear tetrahedra, the recommended value is 3.

`load_division_int`
Should division from file '`partition_l1.ES`' be used? (0 - partition is created in the solver, 1 - partition is read) If partition is read, the file contains for each element, number of subdomain it belongs to. Begins from 1.

3.3 bddcml_upload_subdomain_data

C interface

`void bddcml_upload_subdomain_data(int *nelem, int *nnod, int *ndof, int *ndim, int *meshdim, int *isub, int *nelems, int *nnods, int *ndofs, int *inet, int *linet, int *nnet, int *lnnet, int *nndf, int *lnndf, int *isngn, int *lisngn, int *isvgvn, int *lisvgvn, int *isegn, int *lisegn, double *xyz, int *lxyz1, int *lxyz2, int *ifix, int *lifix, double *fixv, int *lfixv, double *rhs, int *lrhs, int *is_rhs_complete, double *sol, int *lsol, int *matrixtype, int *i_sparse, int *j_sparse, double *a_sparse, int *la, int *is_assembled_int)`

Description

If distribution of data into subdomains exists already in the user application, data should be loaded into the solver using this routine. It may be called repeatedly by each process if more than one subdomain are assigned to that process. It loads the local mesh of the subdomain and assembled subdomain matrix in the coordinate format. Most data are localized to subdomain.

If partitionining into subdomains does not exist in user's application, routine `bddcml_upload_global_data` should be preferred.

Parameters

`nelem` GLOBAL number of elements

`nnod` GLOBAL number of nodes

`ndof` GLOBAL number of degrees of freedom, i.e. size of matrix

`ndim` number of space dimensions

`meshdim` mesh dimension. For 3D elements = `ndim`, for 3D shells = 2, for 3D beams = 1

`isub` GLOBAL index of subdomain which is loaded

`nelems` LOCAL number of elements in subdomain

`nnods` LOCAL number of nodes in subdomain mesh

`ndofs` LOCAL number of degrees of freedom in subdomain mesh

`inet` LOCAL array with Indices of Nodes on ElemenTs - this defines connectivity of the subdomain mesh.

`linet` length of array `inet`. It is given as a sum of entries in array `nnet`.

`nnet` LOCAL array with Number of Nodes on ElemenTs. For each element, it gives number of nodes it is connected to. This is important to locate element entries in array `inet`

`lnnet` length of array `nnet`. It is equal to `nelems`.

`nndf` LOCAL array with Number of Nodal Degrees of Freedom. For each node, it gives number of attached degrees of freedom.

`lnndf` length of array `nndf`. It is equal to `nnods`.

`isngn` array of Indices of Subdomain Nodes in Global Numbering (local to global map of nodes). For each local node gives the global index in original mesh.

`lisngn` length of array `isngn`. It is equal to `nnods`.

`isvgvn` array of Indices of Subdomain Variables in Global Variable Numbering (local to global map of variables). For each local degree of freedom gives the global index in original matrix.

`lisvgvn` length of array `isvgvn`. It is equal to `ndofs`.

`isegn` array of Indices of Subdomain Elements in Global Numbering (local to global map of elements). For each subdomain element gives global number in original mesh.

`lisegn` length of array `isegn`. It is equal to `nelems`.

`xyz` LOCAL array with coordinates of nodes as one array (all X, all Y, all Z) or as two-dimensional array in Fortran (X | Y | Z). Rows are defined by nodes, columns are defined by dimension.

`lxyz1,lxyz2`
dimensions of array `xyz`. In C, lenght of `xyz` is defined as `lxyz1 * lxyz2`. In Fortran, dimension of `xyz` is used as `xyz(lxyz1,lxyz2)`. The `lxyz1` is equal to `nnods`. The `lxyz2` is equal to `ndim`.

`ifix` LOCAL array of Indices of FIXed variables - all dofs with Dirichlet boundary condition are marked with its number, i.e. non-zero entries determine fixed degrees of freedom.

`lifix` length of array `ifix`, equal to `ndofs`.

`fixv` LOCAL array of FIXed Variables - where `ifix` is non-zero, `fixv` stores value of Dirichlet boundary condition. Where `ifix` is zero, corresponding value in `fixv` is meaningless.

`lfixv` length of array `fixv`, equal to `ndofs`.

`rhs` LOCAL array with Right-Hand Side. Values at nodes repeated among subdomains are copied and not weighted.

`lrhs` length of array `rhs`, equal to `ndofs`.

`is_rhs_complete`
is the subdomain right-hand side complete?

- 0 - no, e.g. if only local subassembly of right-hand side was performed - interface values are not fully assembled, solver does not apply weights
- 1 - yes, e.g. if local right-hand side is a restriction of the global array to the subdomain - interface values are complete and repeated for more subdomains, solver applies weights to handle multiplicity of these entries

`sol` LOCAL array with initial SOLution guess. This is used as initial approximation for iterative method.

`lsol` length of array `sol`, equal to `ndofs`.

`matrixtype`
Type of the matrix. This parameter determines storage and underlying direct method of the `MUMPS` solver for factorizations. Matrix is loaded in coordinate format by three arrays described below. Options are

- 0 unsymmetric - whole matrix is loaded
- 1 symmetric positive definite - only upper triangle of the matrix is loaded
- 2 general symmetric - only upper triangle of the matrix is loaded

`i_sparse` array of row indices of non-zero entries

`j_sparse` array of column indices of non-zero entries

`a_sparse` array of values of non-zero entries

`la` length of previous arrays `i_sparse`, `j_sparse`, `a_sparse` (equal to number of non-zeros if the matrix is loaded already assembled)

`is_assembled_int`
is the matrix assembled? The solver comes with fast assembly routine so the users might want to pass just unassembled matrix for each subdomain (i.e. copy of element matrices equipped with global indexing), and let the solver assemble it.

- 0 - no, it can contain repeated entries, will be assembled by solver
- 1 - yes, it is sorted and does not contain repeated index pairs

3.4 bddcml_setup_preconditioner

C interface

```
void bddcml_setup_preconditioner( int *matrixtype, int *use_defaults_int, int
*parallel_division_int, int *use_arithmetic_int, int *use_adaptive_int );
```

Description

Calling this function prepares internal data of the preconditioner. Local factorizations are performed for each subdomain at each level and also the resulting coarse problem on the final level is factored. This might be quite costly routine. Once the preconditioner is set-up, it can be reused for new right hand sides (if the matrix is not changed) by calling `bddcml_upload_subdomain_data` followed by `bddcml_solve`.

Parameters

`matrixtype`
: Type of the matrix. This parameter determines storage and underlying direct method of the `MUMPS` solver for factorizations. Should keep the value inserted to `bddcml_upload_subdomain_data`. Options are

  - 0 unsymmetric - whole matrix is loaded
  - 1 symmetric positive definite - only upper triangle of the matrix is loaded
  - 2 general symmetric - only upper triangle of the matrix is loaded

`use_defaults_int`
: If `> 0`, other options are ignored and the solver uses default options.

`parallel_division_int`
: If `> 0`, solver will use `ParMETIS` to create division on first level. This option is only used for global input (`bddcml_upload_global_data`) and only applies to the first level. Otherwise, `METIS` is used. Default is `1`.

`use_arithmetic_int`
: If `> 0`, solver will use continuity of arithmetic averages on faces in 2D and faces and edges in 3D to form the coarse space. Default is `1`.

`use_adaptive_int`
: If `> 0`, solver will use adaptive averages on faces in 2D and faces in 3D. This might be costly and should be used for very ill-conditioned problems. A generalized eigenvalue problem is solved at each face and weighted averages are derived from eigenvectors. For solving individual eigenproblems, `BLOPEX` package is used. Default is `0`.

3.5 bddcml_solve

C interface

```
void bddcml_solve( int *comm_all, int *method, double *tol, int *maxit, int
*ndecrmax, int *num_iter, int *converged_reason, double *condition_number);
```

Description

This function launches the solution procedure for prepared data. System is solved either by preconditioned conjugate gradient (PCG) method or by preconditioned stabilized Bi-Conjugate Gradient (BiCGstab) method.

Parameters

`comm_all` global communicator. Should be the same as `comm_init` for `bddcml_init` function.

`method` Krylov subspace iterative method
- -1 - use defaults - `tol`, `maxit`, and `ndecrmax` not accessed, BiCGstab method used by default,
- 0 - use PCG,
- 1 - use BiCGstab.

`tol` desired accuracy of relative residual (default 1.e-6).

`maxit` limit on number of iterations (default 1000).

`ndecrmax` limit on number of iterations with non-decreasing residual (default 30) - used to stop a diverging process.

`num_iter` on output, resulting number of iterations.

`converged_reason`
on output, contains reason for convergence/divergence
- 0 - converged relative residual,
- -1 - reached limit on number of iterations,
- -2 - reached limit on number of iterations with non-decreasing residual.

`condition_number`
on output, estimated condition number (for PCG only).

3.6 `bddcml_download_local_solution`

C interface

```
void bddcml_download_local_solution( int *isub, double *sols, int *lsols )
```

Description

Subroutine for getting local solution, i.e. restriction of solution vector to subdomain (no weights are applied).

Parameters

`isub` GLOBAL index of subdomain

`sols` LOCAL array of solution restricted to subdomain

`lsols` length of array `sols`, equal to `ndofs`.

3.7 `bddcml_download_global_solution`

C interface

`void bddcml_download_global_solution( double *sol, int *lsol )`

Description

This function downloads global solution of the system from the solver at root process.

Parameters

`sol` GLOBAL array of solution

`lsol` length of array `sol`, equal to `ndof`

3.8 bddcml_dotprod_subdomain

C interface

`void bddcml_dotprod_subdomain( int *isub, double *vec1, int *lvec1, double *vec2, int *lvec2, double *dotprod )`

Description

Auxiliary subroutine to compute scalar product of two vectors of lenght of subdomain exploiting interface weights from the solver. This routine is useful if we want to compute global norm or dot-product based on vectors restricted to subdomains. Since interface values are contained in several vectors for several subdomains, this dot product or norm cannot be determined without weights.

Parameters

`isub` GLOBAL index of subdomain

`vec1` LOCAL first vector for dot-product

`lvec1` length of `vec1`

`vec2` LOCAL second vector for dot-product, may be same array as `vec1`

`lvec2` length of `vec2`, should be same as `lvec1`

`dotprod` on exit, returns vec1' * weights * vec2

3.9 `bddcml_finalize`

C interface

`void bddcml_finalize( )`

Description

Finalization of the solver. All internal data are deallocated.

Parameters

This routine currently does not take any arguments.