fspai-1.1 – Factorized Sparse Approximate Inverse Preconditioner

Thomas Huckle
Matous Sedlacek

2011 – 09 – 10

Technische Universität München
Research Unit Computer Science V
Scientific Computing in Computer Science

Contents

1 Abstract

Given a sparse symmetric positive definite matrix A, the FSPAI preconditioner computes a sparse approximate inverse lower triangular matrix $L_M \approx L_A^{-1}$, where L_A is the Cholesky factor of $A = L_A^T L_A$. The matrix $M = L_M L_M^T$ can be used as an approximate inverse preconditioner for iterative solvers, like, for example, PCG.

Based on an initial a priori chosen sparsity pattern, FSPAI automatically captures promising indices and updates the sparsity pattern of the preconditioner according to [4, 5]. The algorithm proceeds until $\|L_A L_M - I\|_2 < \epsilon$. By varying ϵ the user may control the quality and the cost of the preconditioner. A very sparse preconditioner is cheap to compute, but may not lead to significant improvements, whereas L_M becomes too expensive to compute if it becomes too dense.

The sequential version, which can be compiled without MPI, uses an own (P)CG implementation if a solver process is requested. The parallel version uses the highly scalable (P)CG solver from the HYPRE [6] package.

2 fspai-1.1 Usage

2.1 Installation

In its current version of fspai-1.1 , there is no common configure script and no common installation package at the moment. Adapt the `Makefile` in the `src` directory and start the compilation process by typing `make` in the `src` directory to start compiling the sourcecode.

For compiling the sequential version fspai-1.1 requires Lapack [7], BLAS [1], CXSparse [3], and UFconfig [11]. The parallel version additionally requires HYPRE [6] and MPI [9]. To compile for a parallel environment use the compiler flag `ENV_MPI`, i.e., compile with `-DENV_MPI`.

2.2 How to run fspai-1.1

- **Sequential version:**

  ```
  ./fspai-1.1 <matrix.mtx> <pattern.mtx|-diag|-L> [options]
  ```

- **Parallel (MPI) version:**

  ```
  mpirun -np <n> ./fspai-1.1 <matrix.mtx> <pattern.mtx|-diag|-L> [options]
  ```

The **mandatory parameters** are:

`matrix.mtx`	The SPD system matrix in *Matrix Market* [8] format.
`pattern.mtx|-diag|-L`	Path to start pattern file in *Matrix Market* format or `-diag` for generating a diagonal start pattern automatically without any file or `-L` for generating a start pattern which prescribes the lower triangular pattern of A itself for the preconditioner. No file required in these two cases.
`n`	Number of processors to use in MPI environment for a parallel computation of L_M.

For getting help on the shell use the `-h` option, e.g., `./fspai-1.1 -h`.

2.3 Options

Optional parameters can be used after the mandatory input in any order, always `-optional_parameter <number|path>`.

Optional parameters are:

`-h(elp)`	**Parameter descriptions to shell** *Usage:* `./fspai-1.1 -h(elp)` *Default:* Don't display help Print the fspai-1.1 usage and optional parameters to shell.
`-ep`	ϵ **tolerance** *Default:* 0.4 *Choice:* `ep` ≥ 0.0 `ep` controls the quality of the approximation of L_M to the inverse of L_A. Lower values of `ep` lead to more computational work, more fill-in and usually to better preconditioners.
`-ns`	**Max. number of improvement steps per column** *Default:* 5 *Choice:* `ns` ≥ 0 Every column of the inverse L_A^{-1} is approximated in several improvement steps (pattern updates). The quality of the approximation is determined by ϵ. If the approximation could not reach the accuracy of ϵ after `ns` steps, fspai-1.1 will use the best approximation so far.

`-mn`

Max. number of new nz candidates per step
Default: 5
Choice: `mn` ≥ 0
For each improvement step there will be chosen `mn` new indices for the current pattern to compute a better approximation in the next step. If no new candidates can be found, the iteration aborts and the approximation will finish.

`-hs`

Hashtable size
Default: 6
Choice: `hs` $\in \{0, 1, 2, 3, 4, 5, 6\}$
fspai-1.1 uses a hashtable to cache remote messages and avoid redundant process communication. It is recommended to use larger hashtables to avoid the linear rehashing mechanism. `-hs 0` forces to switch the hashtable off – no communication will be cached locally.
`-hs 0` Don't use any hashtable.
`-hs 1` Hashtable has size 101.
`-hs 2` Hashtable has size 503.
`-hs 3` Hashtable has size 2503.
`-hs 4` Hashtable has size 12503.
`-hs 5` Hashtable has size 62501.
`-hs 6` Hashtable has size 104743.
This parameter is not available in the sequential version.

`-wp`

Write preconditioner L_M to file
Default: 1 (true)
Choice: `wp` $\in \{0, 1\}$
Whether to write the computed preconditioner into a file or not. The preconditioner will be written in *Matrix Market* format into the specified output file within the current working directory. Default output file is `precond.mtx`.

`-um`

Use mean value
Default: 1 (true)
Choice: `um` $\in \{0, 1\}$
Whether to use a mean bound value for augmenting indices during an improvement step or not. During each improvement step new nz candidates will be computed for a better approximation. With this mean value the number of indices will be reduced to get only few candidates with best improvement. Using this recommended option will reduce the computational costs for constructing the FSPAI.

`-out`	**Specify output file** *Default:* `precond.mtx` *Choice:* `out` be any string containing alphanumeric characters Where to write the computed preconditioner. The preconditioner will be written in *Matrix Market* format to a file named by the specified output string in the current working directory.
`-sol`	**Solver to use** *Default:* 1 *Choice:* `sol` $\in \{0, 1, 2, 3\}$ Which solver is to be used. In a sequential environment only `sol` $\in \{0, 1, 2\}$ is possible. Note, that a right-hand side has to be defined using the `rhs` option. `-sol 0` Don't invoke (P)CG to solve system. `-sol 1` Use PCG solver with computed FSPAI preconditioner. `-sol 2` Perform CG without preconditioner and PCG using computed FSPAI. `-sol 3` Perform PCG with computed FSPAI and PCG using ParaSails preconditioner. The ParaSails preconditioner will be constructed via the HYPRE package automatically.
`-sol_tol`	**Solver tolerance** *Default:* 10^{-6} *Choice:* `sol_tol` ≥ 0.0 `sol_tol` controls the quality/accuracy of the solution computed by the (P)CG solver.
`-sol_maxit`	**Max. number of solver iterations** *Default:* 1000 *Choice:* `sol_maxit` ≥ 0 `sol_maxit` defines the maximum number of iterations (P)CG will perform to compute the solution.
`-sol_out`	**Output file for (P)CG solution** *Default:* `solution.mtx` *Choice:* `sol_out` be any string containing alphanumeric characters Where to write the computed (P)CG solution. The solution will be written in *Matrix Market* format to a file named by the specified output string in the current working directory.
`-rhs`	**Right-hand side for solver** *Default:* 1 `Choice: rhs` $\in \{0, 1\}$

	Specifying the right-hand side for linear system which is to be solved with (P)CG. `-rhs 0` Use random vector as right-hand side. If multiple solution processes are requested, i.e., `sol` > 1 is used, it is guaranteed that the same rhs is used in all solver requests. `-rhs 1` Use ones vector as right-hand side.
`-para_levels`	**Number of levels for ParaSails preconditioner** *Default:* 1 *Choice:* `para_levels` ≥ 0 When constructing ParaSails preconditioner with the HYPRE package, this option specifies the number of levels to be used. See [2] for more details on this parameter.
`-para_thresh`	**Threshold parameter for ParaSails preconditioner** *Default:* 0.1 *Choice:* `para_thresh` $\in [0.0; 1.0]$ When constructing ParaSails preconditioner with the HYPRE package, this option specifies the threshold to be used. See [2] for more details on this parameter.
`-para_filter`	**Filter parameter for ParaSails preconditioner** *Default:* 0.05 *Choice:* `para_filter` $\in [0.0; 1.0]$ When constructing ParaSails preconditioner with the HYPRE package, this option specifies the post-thresholding filter parameter to be used. See [2] for more details on this parameter.
`-alg`	**Fspai algorithm to use for computing preconditioner** *Default:* 0 *Choice:* `alg` $\in \{0, 1, 2, 3\}$ It is possible to invoke different algorithmic branches in fspai-1.1 . For some special matrices `alg` > 0 may speedup the computation. `alg 0` use LAPACK to solve subsystems `alg 1` use CXSparse to solve subsystems `alg 2` use Fspai Caching and LAPACK `alg 3` use Fspai Hashing and LAPACK Fspai Caching uses a buffer of fixed size to store computed Cholesky decompositions. Fspai Hashing uses a hash table. Note that this hash table has nothing in common with the option `-hs`.
`-cs`	**Cache size for Fspai Caching** *Default:* 0

Choice: `cs` ≥ 0
If Fspai Caching is requested via `-alg 2`, a positive cache size greater than zero is necessary.

2.4 Usage examples

- Starting sequential computation of a FSPAI for the matrix *nos3* [8]. Based on a diagonal start pattern the system matrix will be approximated inversely. With an ϵ tolerance of 10^{-3} there will be a maximum of 10 improvement steps, within each a maximum of 3 new candidates will be added to the current pattern without using a mean value bound. Only the preconditioner is to be computed and no solver is to be invoked.

```
./fspai-1.1 /Matrices/nos3.mtx -diag -ep 0.001 -ns 10 -mn 3 -um 0 -sol 0
```

- Starting parallel compuation of a FSPAI on three processors to compute the preconditioner for the matrix *nos3*. Start pattern is the lower triangular sparsity pattern of the system matrix. With an ϵ tolerance of 10^{-2} and a maximum of 5 improvement steps, within each a maximum of 5 new candidates using the mean value bound will be added to the current pattern, the system matrix will be approximated inversely. Both an unpreconditioned CG and a PCG using the computed FSPAI is requested to solve the linear system. The solver tolerance is set to 10^{-7} with a maximum number of 800 iterations. The right-hand side for the linear system is the default one, i.e., a vector of all ones. The Fspai Caching algorithm is to be used with a fixed cache size of 50.

```
mpirun -np 3 opt01 opt02 opt03 ./fspai-1.1 /Matrices/nos3.mtx -L -ep 0.01
-ns 5 -mn 5 -sol 2 -sol_tol 0.0000001 -sol_maxit 800 -alg 2 -cs 50
```

- Starting parallel computation on five processors to compute the FSPAI for the matrix *bcsstk16.mtx* [8]. Based on a specific start pattern from file, the system matrix will be approximated inversely. A static FSPAI without any improvement steps is invoked and both the PCG using FSPAI and PCG using a ParaSails preconditioner are requested to solve the linear system containing a right-hand side of all zeros. For the construction of the ParaSails preconditioner the number of levels is set to 2 and the threshold parameter is switched off with 0.0.

```
mpirun -np 5 opt01 opt02 opt03 opt04 opt05 ./fspai-1.1
/Matrices/bcsstk16.mtx /Matrices/startpattern.mtx -ns 0 -sol 3 -para_levels
2 -para_thresh 0.0 -rhs 0
```

3 FAQ

3.1 What does FSPAI stand for?

FSPAI stands for Factorized SParse Approximate Inverse. The FSPAI algorithm is the factorized analogy to the well-known SPAI algorithm. [4, 5]

3.2 Why does fspai-1.1 taking forever to compute the preconditioner?

Make sure your ϵ tolerance is not too small. You could start with a larger value (e.g., `-ep 0.3`) and reduce it progressively until total execution time starts to increase again.

3.3 Which parameter setting is a good trade-off between performance and accuracy of the approximation?

In general it depends on the system/problem to be preconditioned/solved. However, our experiments revealed that augmenting with more indices in several update steps is more efficient. Depending on sparsity, structure, size, and the initial pattern, update settings can be for instance `ns` $= 10$ with `mn` $= 3$ or `ns` $= 5$ with `mn` $= 5$. A tolerance of `ep` $= 0.3$ or `ep` $= 0.4$ can suffice to get a satisfying quality.

3.4 Does FSPAI always work?

In principle, yes. Unlike many other preconditioners, FSPAI cannot break down if the matrix A is non-singular. Moreover, if one keeps reducing ϵ and use `-mn 1`, fspai-1.1 will eventually compute the exact inverse L_A^{-1}. This may take a very long time.

3.5 When I reduce ϵ the preconditioner does not really improve, why?

Try to increase the number of improvement steps `-ns` and/or the number of indices to be added during one step `-mn`. If that does not help, try to switch off the mean value bound, i.e., use `-um 0`.

3.6 Caching does not improve the computation time, why?

It is advised to use the caching algorithm only when the system matrix is very structured (band, block, etc...). Try to make the cache larger (e.g. 500) when A is supposed to have many different submatrices. Otherwise a small cache of size 50 should be sufficient.

3.7 Which dimension do the system matrices may have?

There is no restriction to the number of columns of A. fspai-1.1 was tested with lots of matrices of different dimension and density. The largest matrices had approximately $1.6 \cdot 10^7$ columns and rows. Note that the system matrices must be square.

3.8 Which input matrices fspai-1.1 was tested with?

Besides matrices generated on our own, like, for example Laplace matrices, fspai-1.1 was tested for various matrices from Matrix Market [8] and from the University of Florida Sparse Matrix Collection [10].

3.9 On which architectures fspai-1.1 was tested on and how many processors have been used so far?

See the website for the environments fspai-1.1 was tested on. Depending on the cluster, fspai-1.1 was used to solve preconditioned systems with more than 8000 processors. We observed high scalability for all our runtime results.

4 Links

4.1 Contacts

- Thomas Huckle

  Department of Computer Science - Research Unit V
  Technische Universität München
  Boltzmannstr. 3
  85748 Garching bei München
  Germany

  ☎ +49-(0)89/289 18 609
  FAX +49-(0)89/289 18 607
  ✉ huckle@informatik.tu-muenchen.de
  www http://www5.in.tum.de/wiki/index.php/Univ.-Prof._Dr._Thomas_Huckle

- Matous Sedlacek

  Department of Computer Science - Research Unit V
  Technische Universität München
  Boltzmannstr. 3
  85748 Garching bei München
  Germany

  ☎ +49-(0)89/289 18 613
  FAX +49-(0)89/289 18 607
  ✉ sedlacek@in.tum.de
  www http://www5.in.tum.de/wiki/index.php/Matous_Sedlacek

4.2 Used tools

The sequential compilation uses:

- **BLAS**: Basic Linear Algebra subroutines [1].

- **LAPACK**: Linear Algebra Package [7].
- **CXSparse**: Extended version of CSparse – Concise Sparse Matrix package [3].
- **UFconfig**: Required by CXSparse [11].

The parallel compilation additionally uses:

- **HYPRE**: High Performance Preconditioners [6].
- **MPI**: Message Passing Interface [9].

4.3 References

References

[1] ATLAS - Automatically tuned linear algebra software. `http://math-atlas.sourceforge.net/`.

[2] E. Chow, *A priori sparsity patterns for parallel sparse approximate inverse preconditioners*, SIAM J. Sci. Comput., 21 (2000), pp. 1804–1822.

[3] CXSparse - Extended Concise Sparse Package. `http://www.cise.ufl.edu/research/sparse/CXSparse/`.

[4] T. Huckle, *Factorized sparse approximate inverses for preconditioning and smoothing*, Selçuk J. of Appl. Math., 1(1) of Sonderheft for 60th Birthday of Prof. Dr. Christoph Zenger (2000), pp. 63–70.

[5] ——, *Factorized sparse approximate inverses for preconditioning*, The Journal of Supercomputing, 25 (2003), pp. 109–117.

[6] HYPRE - High Performance Preconditioners. `http://acts.nersc.gov/hypre`.

[7] LAPACK - Linear Algebra PACKage. `http://www.netlib.org/lapack/index.html`.

[8] Matrix Market. `http://math.nist.gov/MatrixMarket/`.

[9] MPI - Message Passing Interface. `http://www-unix.mcs.anl.gov/mpi/`.

[10] The University of Florida Sparse Matrix Collection. `http://www.cise.ufl.edu/research/sparse/matrices/`.

[11] UFconfig. `http://www.cise.ufl.edu/research/sparse/UFconfig/`.