SPAI – SParse Approximate Inverse Preconditioner

Marcus Grote * Michael Hagemann †

March 16, 2006

Contents

*Department of Mathematics, University of Basel, Rheinsprung 21, 4051 Basel, Switzerland, `Marcus.Grote@unibas.ch`.

†Department of Computer Science, University of Basel.

1 Introduction

Given a sparse matrix A the SPAI Algorithm computes a sparse approximate inverse M by minimizing $\|AM - I\|$ in the Frobenius norm. The approximate inverse is computed explicitly and can then be applied as a preconditioner to an iterative method. The sparsity pattern of the approximate inverse is either fixed a priori or captured automatically:

Fixed sparsity The sparsity pattern of M is either banded or a subset of the sparsity pattern of A.

Adaptive sparsity The algorithm proceeds until the 2-norm of each column of $AM - I$ is less than *eps*. By varying *eps* the user controls the quality and the cost of computing the preconditioner. Usually the optimal *eps* lies between 0.5 and 0.7.

A very sparse preconditioner is very cheap to compute but may not lead to much improvement, while if M becomes rather dense it becomes too expensive to compute. The optimal preconditioner lies between these two extremes and is problem and computer architecture dependent.

The approximate inverse M can also be used as a robust (parallel) smoother for (algebraic) multi-grid methods. For further details please refer to [7, 8, 9].

2 News

Please see the `NEWS` file for a detailed list of changes. Some notable improvements over version 3.1 include:

- Simplified installation procedure (`autoconf` based)
- Threshold based construction of preconditioner, with new MATLAB interface function `spaitau`.
- Banded structure of preconditioner, with new MATLAB interface function `spaidiags`.

3 Installation

3.1 For the impatient

The following gives you the basic commands for a quick build of SPAI, and illustrates some simple ways to get you started with using SPAI as a preconditioner:

Step 1: Unpack the SPAI package, and change to the respective directory:

```
gunzip spai-3.2.tar.gz
tar xvf spai-3.2.tar
cd spai-3.2
```

Step 2: Run the following sequence and keep your fingers crossed:

```
configure --with-matlab
make
make check
```

(Please note that a working FORTRAN compiler is required to detect the system calling conventions. Furthermore, MATLAB needs to be in the current `$PATH` in order to be detected and used. See Section 3.2.3 for further configuration options on various platforms.)

Step 3: Test the command line interface:

```
cd src
./spai ../data/m1.mm
./spai ../data/m1.mm -sc 1 -ta 0.7
```

(Please note that the matrix file needs to be the first parameter.)

Step 4: Test the MATLAB interface (if available):

```
cd ../matlab
matlab
>> help spai
>> % Load A as matrix 'm1.dat' and create RHS b=A*1.
>> A = spconvert(load('m1.dat'));
>> b = A * ones(size(A,1), 1);

>> % Solve Ax=b, without preconditioning
>> x = bicgstab(A, b, 1e-8, 200);

>> % Compute adaptive SPAI preconditioner (SPAI 3.1)
>> M = spai(A);
>> spy(M);

>> % Solve Ax=b, with SPAI preconditioning
>> x = bicgstab(A, b, 1e-8, 200, @(y) M*y);

>> % Compute block SPAI preconditioner (block size=3)
>> M = spai(A, 0.6, 5, 5, 3, 100000, 0);

>> % Compute threshold based SPAI preconditioner
>> M = spaitau(A, 0.8, 0);

>> % Compute banded SPAI preconditioner
>> M = spaidiags(A, 5, 5, 0);
```

See Figure 1 for a comparison of the sparsity patterns for the above Matlab function calls, and the convergence history of a preconditioned BICGSTAB iteration.

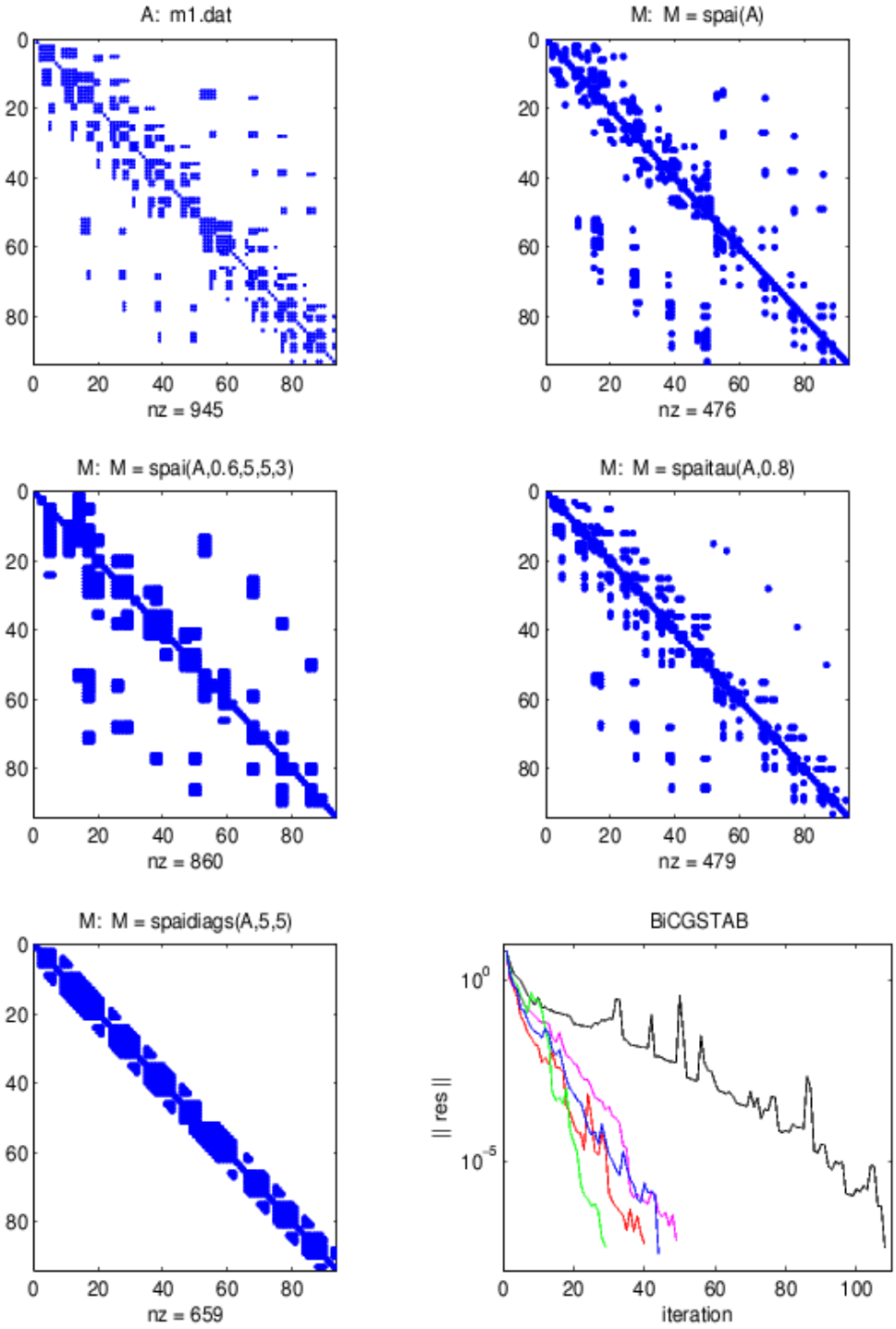

Figure 1: Comparison of sparsity patterns for different SPAI Matlab functions, and convergence history of preconditioned BICGSTAB iteration.

3.2 Detailed installation instructions

3.2.1 Using configure

The '`configure`' shell script attempts to guess correct values for various system-dependent variables used during compilation. It uses those values to create a `Makefile` in each directory of the package. It may also create one or more `.h` files containing system-dependent definitions. Finally, it creates a shell script `config.status` that you can run in the future to recreate the current configuration, a file `config.cache` that saves the results of its tests to speed up reconfiguring, and a file `config.log` containing compiler output (useful mainly for debugging `configure`).

The simplest way to compile this package is:

1. `cd` to the directory containing the package's source code and type `./configure` to configure the package for your system. If you're using `csh` on an old version of System V, you might need to type `sh ./configure` instead to prevent `csh` from trying to execute `configure` itself.

   Running `configure` takes a while. While running, it prints some messages telling which features it is checking for.

2. Type `make` to compile the package.

3. Optionally, type `make check` to run any self-tests that come with the package.

4. Type `make install` to install the programs and any data files and documentation.

5. You can remove the program binaries and object files from the source code directory by typing `make clean`. To also remove the files that `configure` created (so you can compile the package for a different kind of computer), type `make distclean`.

3.2.2 Optional Features

`configure` accepts the following command line options:

`--help` Print a summary of the options of `configure`, and exit.

`--with-blas={no|"-llibrary1 -llibrary2"}` Override the default option for BLAS libraries. By default, various known and vendor specific BLAS libraries are searched for and tested. This search can be overridden by specifying the BLAS linker options.

Example: To build with a free ATLAS BLAS installed in a non-standard place, try the following:

```
./configure --with-blas="-L/opt/atlas -lf77blas -lcblas -latlas"
```

If no suitable system BLAS library is found, a local library is built. This will typically decrease the performance of `spai`.

`--with-lapack={no|"-llibrary1 -llibrary2"|/path/to/lapack-libary.a}` Override the default option for LAPACK libraries. By default, various known and vendor specific LAPACK libraries are searched for and tested. This search can be overridden by specifying the LAPACK linker options.

If no suitable system LAPACK library is found, a local library is built. This will typically decrease the performance of `spai`.

`--with-matlab` Checks for a MATLAB installation and compiles the MATLAB interface. MATLAB must be in the current `$PATH`.

`--with-MPI={lam|mpich|generic}` Compiles a parallel version using an already installed version of the MPI library.

The value `lam` configures the package using specific details of the LAM implementation of MPI. See the *LAM homepage* http://www.lam-mpi.org/ for details on how to obtain, install and run LAM.

The value `mpich` configures the package using specific details of the MPICH implementation of MPI. See the *MPICH homepage* http://www-unix.mcs.anl.gov/mpi/mpich/ for details on how to obtain, install and run MPICH.

The value `generic` tries to guess the needed libraries and binaries needed.

3.2.3 Compilers and Options

Some systems require unusual options for compilation or linking that the `configure` script does not know about. You can give `configure` initial values for variables by setting them in the environment. Using a Bourne-compatible shell, you can do that on the command line like this:

```
CC=c89 CFLAGS=-O2 LIBS=-lposix  ./configure
```

Or on systems that have the `env` program, you can do it like this:

```
env CC=c89 CFLAGS=-O2 LIBS=-lposix  ./configure
```

We have successfully installed and tested SPAI on the following systems:

- **i686-pc-linux-gnu** with GNU `gcc`

  All features are typically discovered automatically. A working FORTRAN compiler needs to be installed. Tested for `libspai.a` library, CLI, Matlab-interface, and MPI (LAM).

- **sparc-sun-solaris2.8** with SUN `cc` and SUN high performance library

  We used the following options to avoid linking problems due to the OPENMP stubs.

  ```
  CC=cc F77=f90 CFLAGS="-xopenmp=stubs"  ./configure
  ```

  The `sunperf` libraries are detected automatically. If you want to compile the MATLAB interface, you cannot link to the `sunperf` libraries. A possible solution is to compile the local BLAS and LAPACK libraries with:

  ```
  CC=cc F77=f90 CFLAGS="-xopenmp=stubs"  ./configure  \
        --with-matlab --with-blas=no --with-lapack=no
  ```

- **x86_64-unknown-linux-gnu**

  We use the AMD `ACML` library:

```
./configure --with-blas=/opt/acml/gnu64/lib/libacml.a
```

With MATLAB, the following options were necessary, in order to make the library dynamically linkable:

```
CFLAGS=-fpic ./configure --with-matlab  \
                         --with-blas=/opt/acml/gnu64/lib/libacml.a
```

- **powerpc-apple-darwin8.2.0** (with gcc)

  All features are typically discovered automatically. Tested for library, and CLI.

3.3 Installation

Automatic installation is available through the `automake` package (`make install`), but we recommend a local installation. If the library needs to be accessible to multiple users, the `libspai.a` file can be copied to an appropriate location (`/usr/local/lib`).

3.3.1 Matlab interface

If you want to use the MATLAB interface outside of the `spai-3.2/matlab` directory, the simplest way is to include the directory in the `$MATLABPATH` environment variable.

Inside MATLAB, this is possible with the `addpath` and `path` functions:

- Prepend to `MATLABPATH`:

```
>> addpath /dir1 /dir2
```

- Append to `MATLABPATH`:

```
>> p = path()
>> path (p, '/dir1')
```

You can include these commands in the `/matlab/startup.m` file, which is automatically loaded every time MATLAB is started. You can also change the path permanently by setting the environment variable `$MATLABPATH` in your shell startup script.

3.3.2 Installed files

If you use the `make install` command for the installation, the following files are installed in the `$PREFIX` directory (the default is `/usr/local/`):

```
$(PREFIX)/bin
$(PREFIX)/bin/spai
$(PREFIX)/bin/convert
$(PREFIX)/lib
$(PREFIX)/lib/libspai.a
$(PREFIX)/matlab
$(PREFIX)/matlab/spai
$(PREFIX)/matlab/spai/spai_full.mexglx
```

```
$(PREFIX)/matlab/spai/spai.m
$(PREFIX)/matlab/spai/spaitau.m
$(PREFIX)/matlab/spai/spaidiags.m
$(PREFIX)/share
$(PREFIX)/share/doc
$(PREFIX)/share/doc/spai
$(PREFIX)/share/doc/spai/spaidoc.ps
$(PREFIX)/share/doc/spai/spaidoc.pdf
```

You can change the `$PREFIX` directory with the `--prefix=` option of the `configure` script.

3.4 Usage

3.4.1 Test program "spai"

The test program is called "spai" and `make install` will install it in your `$prefix/bin` directory. You can set your `PATH` environment variable accordingly.

The program will

1. Read a sparse matrix A
2. Optionally read a rhs vector b (or construct one)
3. Construct a SPAI preconditioner M
4. use BICGSTAB to solve $Ax = b$, using M as a preconditioner.

The serial program is executed with:

```
spai <A> [b] [options]
```

while the parallel version is executed with:

```
mpirun -np <n> ./spai <A> [b] [options]
```

(Items in <> are required; items in [] are optional.)

<A> must be a file in Matrix Market coordinate format. See the file data/m1.mm for an example. (Matrix Market is a repository of sparse matrices on the web. The only Matrix Market format currently supported is "real general". This format is very similar to the MATLAB sparse matrix format. See the *MatrixMarket Homepage* http://math.nist.gov/MatrixMarket/ for details.

[b] is an optional dense vector giving the right-hand-side for the BICGSTAB solver. It must be a file in Matrix Market array format. See the file `data/m1_rhs.mm` for an example. If the RHS is not given then BICGSTAB will will use $A * \mathbb{1}$ as a RHS, where $\mathbb{1}$ is a vector of all ones. The solution of the system $Ax = b$ will be written to the file `solution.mm`.

There are a number of optional parameters. They all have default values and are of the form `-xx s`, where `xx` is a two-letter abbreviation of the parameter's name, and `s` determines its value. Usually only a few need to be modified in practice.

-sc sparsity control; default is 0

0 = adaptive {`-ep`, `-mn`, `-ns`, `-bs` }
1 = specified tau { `-ta` },
2 = fixed diagonals { `-ud`, `-ld` }

These options for sparsity control are mutually exclusive.
The main diagonal is always included.

-ep *eps* parameter for SPAI; default is 0.6

eps must be between 0 and 1. It controls the quality of the approximation of M to the inverse of A. Higher values of *eps* lead to more work, more fill-in, and usually better preconditioners.

-bs block size; default is 1

A block size of 1 treats A as a matrix of scalar elements. A block size of $s > 1$ treats both A and M as $s \times s$ block matrices. A block size of 0 treats A as a matrix with variable sized blocks, which are determined by searching for dense square diagonal blocks in A. This can be very effective for finite-element matrices.

SPAI will convert A to block form, use a block version of the preconditioner algorithm, and then convert the result back to scalar form.

In many cases a block-size parameter $s \neq 1$ can lead to significant improvements in performance.

-ns maximum number of improvement steps per row in SPAI; default is 5

SPAI constructs an approximation to every column of A^{-1} in a series of improvement steps. The quality of the approximation is determined by *eps*. If a residual less than *eps* is not achieved after `ns` steps, SPAI simply uses the best approximation obtained so far.

-mf message file for warning messages; default is 0 (`/dev/null`)

Suppose you are using the option `"-mf foo"`. Whenever SPAI fails to achieve an approximation of epsilon for a column of M it writes a message to the file "fooi" where i is the MPI rank of the processor generating the message (or i=0 in the serial case).

-mn maximum number of new nonzero candidates per step; default is 5

-sp symmetric pattern; default is 0

If A has a symmetric nonzero pattern use `-sp` 1 to improve performance by eliminating some communication in the parallel version.

-mb size of various working buffers in SPAI; default is 100000

Increase this if you run into problems with crashes. 100000 is a very generous size for most problems.

-cs cache size {0,1,2,3,4,5} in SPAI; default is 5 (the biggest cache size)

SPAI uses a hash table to cache messages and avoid redundant communication. We recommend always using `-cs 5`. This parameter is irrelevant in the serial version.

-mi maximum number of iterations in BICGSTAB; default is 500

-to tolerance in BICGSTAB default; is 1.0e-8

-lp left preconditioning; default is 0 → right preconditioning

SPAI stores matrices in a compressed-column format and by default computes a right preconditioner. If the -lp 1 option is given SPAI will use a compressed-row format and compute a left preconditioner.

-bi read matrix as a binary file; default is 0

There is a program called "convert" in the driver directory that converts an ASCII matrix file (in Matrix Market format) to a binary file. This can greatly speed up the input of large matrices.

Try:

```
convert  ../data/m1.mm  m1.binary
spai  m1.binary  -bi 1
```

-vb verbose; default is 1

Print parameters, timings and matrix statistics.

-db debugging level; default is 0

-ld # of lower (below main) diagonals; default is 0

The main diagonal is always included.

-ud # of upper (above main) diagonals; default is 0

-ta (tau) Threshold for non-zero entries in M; default is 0

Only those entries M_{ij} are included for which

$$|A_{ij}| > (1 - tau) * max_j |A_{ij}|, \quad \text{i.e.}$$

$tau = 0$: main diagonal only,
$tau = 1$: full pattern of A.

The main diagonal is always included.

3.4.2 Running `make` in general

Running checks You can run `make check` to perform some automatic checks to see if the libraries and the optional MATLAB interface have been compiled correctly. Numerical codes obviously usually do not execute identically on different machines. Therefore small changes in the computations and consequently in the output do occur. If there really is a difference between the expected output and the output produce by your compilation you will see the difference produce by `diff`. You will most probably see something like this:

```
63c63
< 10 4.020936e-10
---
```

```
> 10 4.020935e-10
65c65
< 12 2.528134e-12
---
> 12 2.528133e-12
check difference in output!!!
```

Anything much worse than this points to an error. Note that the test may still be considered as "passed" by make.

The same might apply for the MATLAB interface, even for differing versions of MATLAB:

```
3,5c3,5
<                    Copyright 1984-2000 The MathWorks, Inc.
<                          Version 6.0.0.88 Release 12
<                                  Sep 21 2000
---
>                    Copyright 1984-2001 The MathWorks, Inc.
>                         Version 6.1.0.450 Release 12.1
>                                  May 18 2001
check difference in output!!!
```

3.4.3 Remarks

The BICGSTAB routine is only intended for testing. It is not efficient in a parallel environment because it uses a sparse matrix-vector multiply that does not scale well. There is a PETSc interface to SPAI, please consult the PETSc documentation.

BICGSTAB may sometimes produces slightly different results when run on different numbers of processors. This is not a bug. The arithmetic operations are done in different order, and can therefore lead to different convergence histories. SPAI always produces the same preconditioning matrix on any number of processors when the `"-bs 1"` option is used. This is because the scalar-valued matrix is distributed before blocking is done and the distribution might break up blocks.

To find good SPAI parameters for your application we suggest that you start with a fairly large value for *eps*, like 0.7, and then decrease *eps* until you find the best setting. Start the -ns parameter as something large, like 20, and use the -mf parameter to see how many columns don't achieve *eps* (if any). It may be most practical to use a value for -ns that results in some columns not achieving *eps*. We also encourage you to try the different -bs options available.

You can write the matrices (either A or M) with a routine called "`write_mm_matrix`". Look for some commented-out code in `test_spai.c` (in driver) to see how this is done.

Finally, if you are modifying the code there is a debugging trick that is very helpful. If you use an option "-db 1" the program will create n files: dbg0, dbg1, ..., where n is the number of processors. If you insert code like the following into your program you can print out information from different processors in these files.

```
if (debug) {
  fprintf(fptr_dbg,...);
  fflush(fptr_dbg);
}
```

See the file `spai.c` in `lib` for an example.

4 FAQ

4.1 Why is SPAI taking forever to compute the preconditioner?

You are probably using too low a value for *eps*. You should start with a relatively large value, say *eps*=0.7 or 0.8, and progressively reduce *eps* until the total execution time starts to increase again. This is usually the value of *eps* for which the time spent to compute the preconditioner is approximately equal to that spent in the iterative solver. Usually the optimal value for *eps* lies between 0.5 and 0.7.

4.2 When I reduce *eps* the preconditioner does not really improve, why?

If you reduce *eps* and the preconditioner does not improve, rerun SPAI with the `-mf my_file` option and check whether any messages have been written to my_file. If any columns in the preconditioner did not satisfy the *eps* criterion, `my_file` will say so, and you need to increase the number of improvement steps with the -ns option (try 5, 10, 20), before decreasing *eps* any further. However, if `my_file` is empty simply proceed in further reducing *eps*.

4.3 What does SPAI stand for?

SPAI stands for SParse Approximate Inverse. The name was invented some time in the Spring of 1994 during a typical lunch outside on the beautiful lawn of the Main Quad at Stanford University: an improvised "déjeuner sur l'herbe" amongst Rodin sculptures below sunny Californian skies. Present were M. Grote, T. Huckle (TU-Munich), and A.-J. van der Veen (TU-Delft).

4.4 Why does the preconditioner tend to get worse as I increase the block size while keeping *eps* fixed?

For the same value of *eps* an increase in the block size often results in a slight deterioration in the quality of the preconditioner; this is usually compensated by a large reduction in computing time. The reason is that the *eps* criterion is more stringent in the scalar (bs = 1) than in the block case. Nevertheless, the overall reduction of $\|AM - I\|_F$ is usually identical.

4.5 Does SPAI always work?

In principle, yes. Unlike many other preconditioners, SPAI cannot break down if the matrix A is non-singular. Moreover, if one keeps reducing *eps*, SPAI will eventually compute the exact inverse. Of course this may take a VERY long time...

4.6 Configure hangs at `mexfileextension`-check. Why?

You probably have the `DISPLAY` variable not set correctly. Configure then hangs trying to start MATLAB. Try starting MATLAB manually. When successful, try `configure` again.

4.7 I still have questions, where do I get further information?

Please direct your questions to the SPAI mailing list under

https://www.maillist.unibas.ch/mailman/listinfo/spai.

5 Authors

The main authors of the SPAI code are Stephen Barnard and Marcus Grote. The configuration system was programmed by Oliver Bröker and Michael Hagemann.

We would also like to thank the following people for using SPAI and helping to improve the package: Victor Eijkhout (PETSc interface), Oliver Ernst, Hans Grote, Jan Hesthaven, Thomas Huckle, Olaf Schenk, Barry Smith, Bora Uçar.

References

[1] A Block Version of the SPAI Preconditioner, S.T. Barnard and M. J. Grote, in Proc. 9th SIAM Conf. on Parall. Process. for Sci. Comp., held March 1999.

[2] A Portable MPI Implementation of the SPAI Preconditioner in ISIS++, S.T. Barnard and R.L. Clay, in Proc. Eighth SIAM Conf. on Parallel Process. for Sci. Comp., held March 1997.

[3] An MPI Implementation of the SPAI Preconditioner on the T3E, S. Barnard, L.M. Bernardo and H.D. Simon, technical report LBNL-40794, 1997.

[4] Parallel Preconditioning with Sparse Approximate Inverses, M.J. Grote and T. Huckle, SIAM J. of Scient. Comput. 18(3), 1997.

[5] Effective Parallel Preconditioning with Sparse Approximate Inverses, M. Grote and T. Huckle, in Proc. Seventh SIAM Conf. on Parallel Process. for Sci. Comp., held February 1995.

[6] Parallel Preconditioning and Approximate Inverses on the Connection Machine, M. Grote and H.D. Simon, in Proc. Sixth SIAM Conf. on Parallel Process. for Sci. Comp., held March 1993.

[7] Sparse approximate inverse smoothers for geometric and algebraic multigrid, O. Bröker and M.J. Grote, Applied Numerical Mathematics 41(1), 2002.

[8] Parallel Multigrid Methods using Sparse Approximate Inverses, O. Bröker, PhD Thesis, Dept. of Computer Science, ETH Zurich, 2003.

[9] Robust Parallel Smoothing for Multigrid Via Sparse Approximate Inverses, O. Bröker, M. Grote, C. Mayer, and A. Reusken, SIAM J. of Scient. Comput. 23(4), 2001.

[10] O. Bröker and M.J. Grote, Parallel Algebraic Multigrid via Sparse Approximate Inverses, in Proc. of 16th IMACS World Congress 2000, held August 2000.