FlexiBLAS - A flexible BLAS library with runtime exchangeable backends

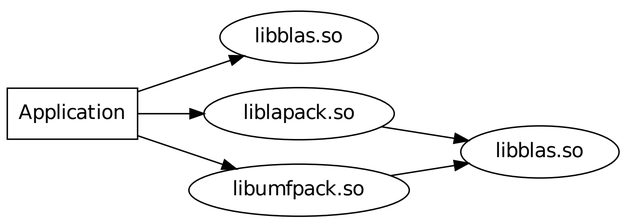
Martin Köhler and Jens Saak

Max Planck Institute for Dynamics of Complex Technical Systems Magdeburg,
Research Group Computational Methods in Systems and Control Theory

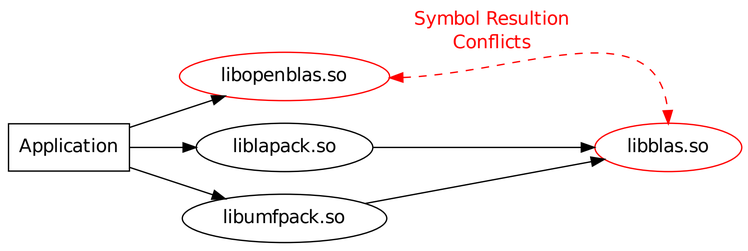
Motivation

The BLAS library is one of the fundamental software packages in Scientific Computing. Beside the reference implementation from Netlib many optimized variants like Intel® MKL, AMD ACML, Apple Accelerate, OpenBLAS or ATLAS exist.

We consider an example application which depends on BLAS, LAPACK and UMFPACK. LAPACK and UMFPACK depend on BLAS as well. If we link an application dynamically against a default BLAS implementation named `libblas.so` we get:



If we now want to exchange the BLAS library for benchmark or debugging purposes and simply relink our application, e.g., against `libopenblas.so`, but leave `liblapack.so` and `libumfpack.so` untouched we end up with:



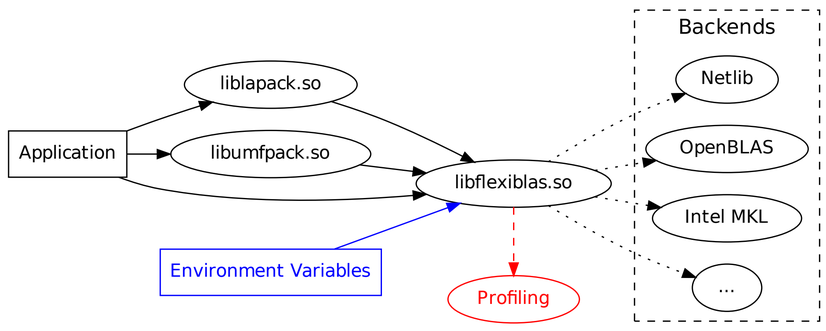
This conflict yields an unusable application or not all parts of the application use the desired BLAS variant.

Our Aim: Find a possibility to exchange the BLAS library at runtime without relinking the application and all its dependencies. Further the mechanism should not require super user privileges like the `update-alternatives` or the `eselect` mechanism.

Key Idea

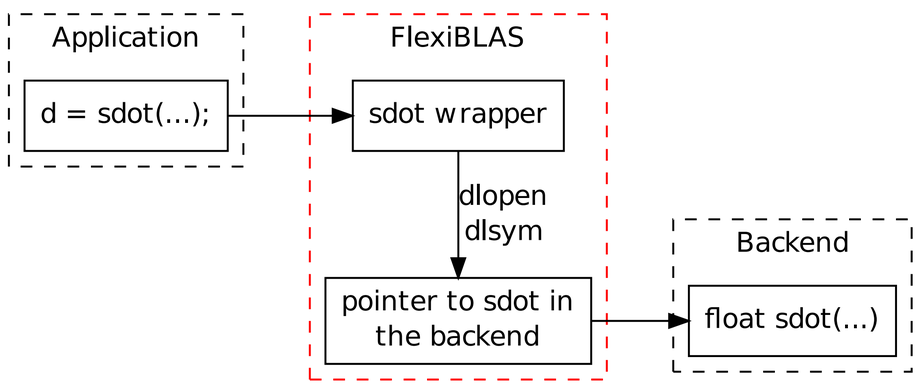
We introduce a *wrapper*-library which passes all BLAS function calls to a previously selected backend library. Our wrapper, i.e. Application Programming Interface (API) and Application Binary Interface (ABI), exactly looks like the Netlib reference BLAS implementation. The backend is selected via an environment variable or a configuration tool which works with standard user-privileges.

Now, we only have to link all libraries against the wrapper library `libflexiblas.so` and we can switch painlessly between different BLAS implementations



Implementation

Our wrapper uses the POSIX `dlopen` mechanism to load the selected BLAS backend. Each BLAS symbol is resolved using `dlsym` when the application starts. These symbols are used to call the real BLAS function from the wrapper functions in our library. Each wrapper function has the same signature as the corresponding Netlib BLAS function but only passes its arguments to the previously loaded symbol. In the case of the single precision dot-product `sdot` this looks like:



Eventually existing return values take the path back to the application again.

Octave Interface

Change the number of threads in the backend:

Many BLAS libraries like OpenBLAS, Intel® MKL, or AMD ACML can adjust the number of threads:

```
flexiblas_set_num_threads(1);
[L, U, P] = lu(A);              % Performed with one thread.

flexiblas_set_num_threads(4);
[Q, R] = qr(A);                 % Performed with four threads.
```

Load and switch backends:

Due to different optimizations, debugging, or benchmarks one can load different BLAS backends and switch between them:

```
n = 4096;
A = rand(n); B = rand(n); C = zeros(n);

% Load OpenBLAS and Intel MKL
backend(1) = flexiblas_load_backend("OPENBLAS");
backend(2) = flexiblas_load_backend("MKL");
if ( any ( backend < 0 ) )
  error('Failed to load backends');
end

for i=1:length(backend)
  flexiblas_switch (backend(i));
  tic;
  C = A * B;
  t = toc;
  fprintf(1, 'Backend %s took %g seconds.\n');
end

flexiblas_switch(0);   % Restore the default backend.
```

Profiling

By introducing a wrapper function for each BLAS function, we get the opportunity to include time measurement and a call counter. The profiling results are printed to the screen or a file after the application terminates. In this way, we can profile the usage of the BLAS in arbitrary applications.

Example: Analyze the BLAS usage during the LU decomposition of a random matrix in GNU Octave:

```
A = rand(2000,2000);
[L,U,P] = lu (A);
```

Function	Runtime in s	Calls
dgemm	2.6776190e+00	31
dger	7.0498466e-02	1968
dscal	1.4417171e-03	1999
dswap	1.4176369e-03	1996
dtrsm	1.9094133e-01	31

Features

- No super user privileges are required to exchange the BLAS library.
- No need to recompile dependent libraries.
- “Painless” usage of complex BLAS libraries like Intel® MKL.
- No complicated dealing with `LD_LIBRARY_PATH` or `LD_PRELOAD` anymore.
- Easy basic profiling without special compiler flags or additional tools.
- Generic interface to the “set-number-of-threads” function (if available).
- BLAS-extensions from OpenBLAS and Intel® MKL are included.
- Automatic adjustment of `ifort/g77`-like complex return values.

In development:

- LAPACK support (≈ 3800 subroutines, code generator nearly ready).
- Backends for accelerator support in level-2 and level-3 BLAS calls.

Resources

- `http://www.mpi-magdeburg.mpg.de/projects/flexiblas/`
- *M. Köhler and J. Saak, FlexiBLAS - A flexible BLAS library with runtime exchangeable backends, LAPACK Working Note 284, 2013*