MAX PLANCK INSTITUTE
FOR DYNAMICS OF COMPLEX
TECHNICAL SYSTEMS
MAGDEBURG

CSC

COMPUTATIONAL METHODS IN
SYSTEMS AND CONTROL THEORY

20 YEARS
1998-2018

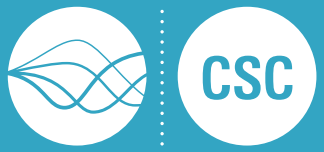
FlexiBLAS

Switching BLAS libraries made easy

Martin Köhler

joint work with Jens Saak, Christian Himpe, and Jörn Papenbroock

January 29, 2018



What is BLAS?

Basic Linear Algebra Subprograms (BLAS)

"The BLAS (Basic Linear Algebra Subprograms) are routines that provide standard building blocks for performing basic vector and matrix operations. . . . Because the BLAS are efficient, portable, and widely available, they are commonly used in the development of high quality linear algebra software, LAPACK for example."[4]

[4] From: `http://www.netlib.org/blas/faq.html` – What and where are the BLAS?

What is BLAS?

BLAS routine organization

Let α, β be scalars, x, y be vectors, A, B, C be matrices.

level	included operations	data	flops
1	αx, $\alpha x + y$, $x^* y$, $\|\|x\|\|_2$, $\|\|x\|\|_1$, $\|\|x\|\|_\infty$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
2	$\alpha Ax + \beta y$, $\alpha A^* x + \beta y$, $A + \alpha xy^*$, $A + \alpha xx^*$, $A + \alpha xy^* + \beta yx^*$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
3	$\alpha AB + \beta C$, $\alpha AB^* + \beta C$, $\alpha A^* B^* + \beta C$, $\alpha AA^* + \beta C$, $\alpha A^* A + \beta C$ rank k updates $\alpha A^* B + \beta C$, $\alpha B^* A + \beta C$ rank $2k$ updates	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$

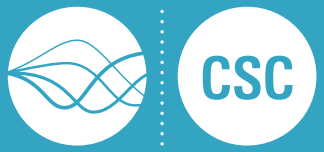
What is BLAS?

BLAS routine organization

Let α, β be scalars, x, y be vectors, A, B, C be matrices.

level	included operations	data	flops
1	αx, $\alpha x + y$, $x^* y$, $\|\|x\|\|_2$, $\|\|x\|\|_1$, $\|\|x\|\|_\infty$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
2	$\alpha Ax + \beta y$, $\alpha A^* x + \beta y$, $A + \alpha xy^*$, $A + \alpha xx^*$, $A + \alpha xy^* + \beta yx^*$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
3	$\alpha AB + \beta C$, $\alpha AB^* + \beta C$, $\alpha A^* B^* + \beta C$, $\alpha AA^* + \beta C$, $\alpha A^* A + \beta C$ rank k updates $\alpha A^* B + \beta C$, $\alpha B^* A + \beta C$ rank $2k$ updates	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$

Level 3 BLAS especially attractive for communication avoidance and parallelism.



What is BLAS?

Some important BLAS implementations

Open Source

- NetLib BLAS: `http://www.netlib.org/blas/` (the reference)
- OpenBLAS: `http://www.openblas.net/` (uses assembler level optimization)
- Automatically Tuned Linear Algebra Software (ATLAS):
  `http://math-atlas.sourceforge.net/`
  (provides automatic compile-time tuning for specific processors and threading)
- BLIS (BLAS-like Library Instantiation Software Framework):
  `https://github.com/flame/blis`
  (alternative approach to BLAS, with wrappers available)

Hardware Vendor Implementations

- Intel® Math kernel library (MKL):
  `http://software.intel.com/en-us/intel-mkl/`
- AMD Core Math Library (ACML): . . . discontinued
- Apple Accelerate, IBM ESSL, . . .

Why do we need yet another BLAS library?

Linker Problems

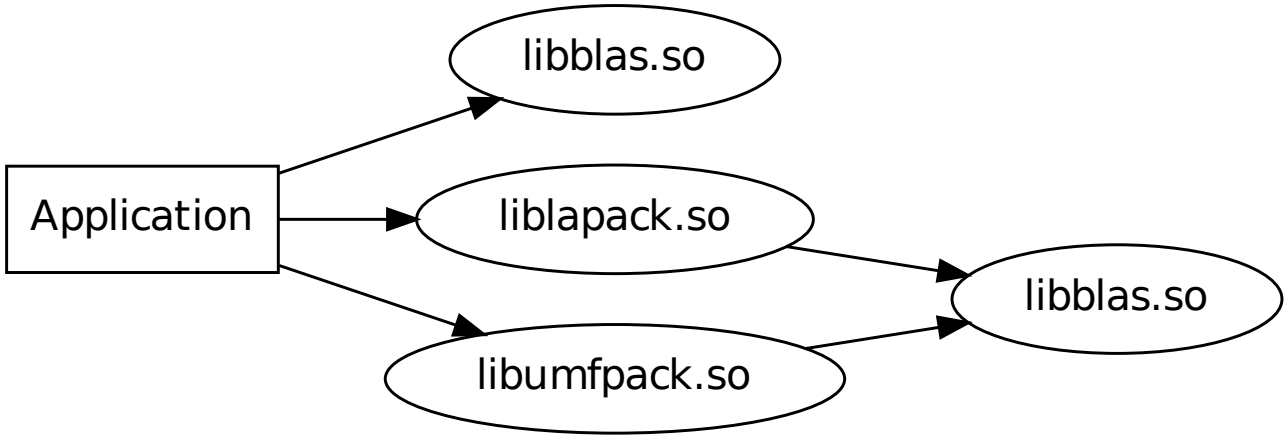


Figure: A sample application using BLAS

```
gcc -o application app.o -lumfpack -llapack -lblas
```

Why do we need yet another BLAS library?

Linker Problems


```
$ ldd ./application
linux-vdso.so.1 => (0x00007ffc2d1de000)
libumfpack.so.5.7.1 => /.../libumfpack.so.5.7.1
liblapack.so.3 => /.../liblapack.so.3
libblas.so.3 => /.../libblas.so.3
libc.so.6 => /.../libc.so.6
...
```

```
gcc -o application app.o -lumfpack -llapack -lblas
```

Why do we need yet another BLAS library?

Linker Problems

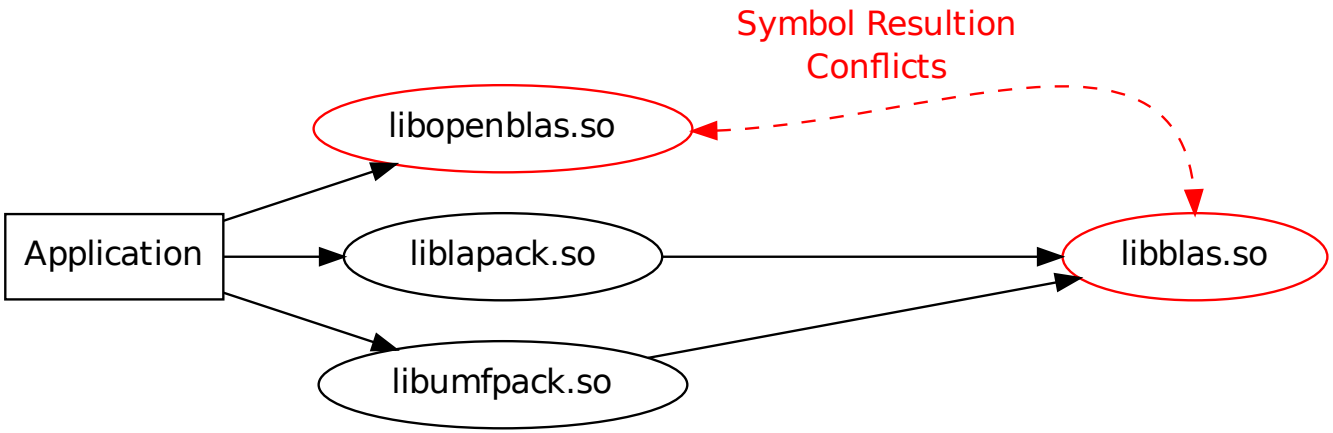
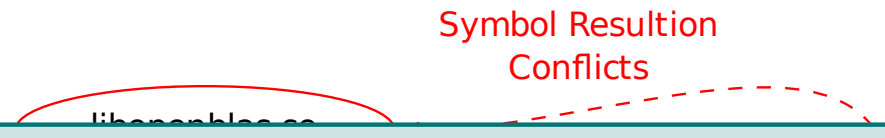


Figure: . . . after linking with a different BLAS-implementation

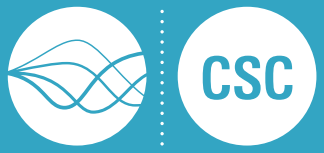
```
gcc -o application app.o -lumfpack -llapack
-lopenblas
```

Why do we need yet another BLAS library?
Linker Problems




```
$ ldd ./application
linux-vdso.so.1 => (0x00007ffc2d1de000)
libumfpack.so.5.7.1 => /.../libumfpack.so.5.7.1
liblapack.so.3 => /.../liblapack.so.3
libopenblas.so.0 => /.../libopenblas.so.0
libc.so.6 => /.../libc.so.6
libm.so.6 => /.../libm.so.6
libblas.so.3 => /.../libblas.so.3
...
```

```
gcc -o application app.o -lumfpack -llapack
    -lopenblas
```



Why do we need yet another BLAS library?

Linker Problems: Existing Solutions

- `LD_LIBRARY_PATH` / `LD_PRELOAD`
  only applicable for single file implementations
  (i.e. NOT Intel® MKL, or ATLAS)
- static libraries
  drastically increased binary sizes, often complicated linking, painful in large projects
- `update-alternatives` (Debian/Ubuntu/Suse)
  requires super-user privileges and has similar restrictions as `LD_LIBRARY_PATH` / `LD_PRELOAD`
- `eselect` / `pkg-config` (Gentoo)
  requires super-user privileges and switches at build-time only
- *BSD ports/pkgsrc/dports
  Links agains libblas.so if already installed otherwise installs some BLAS implementation depending on the maintainer.

Why do we need yet another BLAS library?

Compatibility Issues

`gfortran` vs `g77/intel` interface style

- **different calling sequences:**
  `gfortran` and `g77/f2c/intel` return complex numbers as additional function parameters.
- **affected routines:** `zdotc`, `zdotu`, `cdotc`, `cdotu` (level 1)

Why do we need yet another BLAS library?

Compatibility Issues

`gfortran` vs `g77/intel` interface style

- **different calling sequences:**
  `gfortran` and `g77/f2c/intel` return complex numbers as additional function parameters.
- **affected routines:** `zdotc`, `zdotu`, `cdotc`, `cdotu` (level 1)

auxiliary routine treatment

- Routines `sc/dzabs1` are missing in ATLAS and derived implementations, such as Apple Accelerate / AMD ACML.
- Intel® MKL and OpenBLAS extend the BLAS routine set by: `xAXPBY`, `xOMATCOPY`, ...

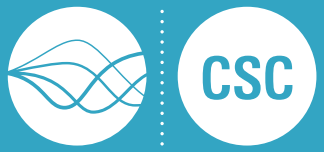
Why do we need yet another BLAS library?

Compatibility Issues

dependency detection problems

Correct/reliable detection of alternative BLAS implementations not guaranteed for many software packages:

- faulty `autotools` scripts,
- old CMake versions,
- hard-coded library names,
- non-standard library locations.



Why do we need yet another BLAS library?

Profiling

- Profiling usually requires additional compiler settings,
- Profiler data requires additional (sometimes confusing) tools for evaluation,
- Profilers often induce considerable overhead influencing the runtime behavior of the profiled application,
- Profiling needs to be active for entire applications.

Why do we need yet another BLAS library?

Profiling

- Profiling usually requires additional compiler settings,
- Profiler data requires additional (sometimes confusing) tools for evaluation,
- Profilers often induce considerable overhead influencing the runtime behavior of the profiled application,
- Profiling needs to be active for entire applications.

Often only execution times and numbers of calls of single routines are of interest.

Our Solution – FlexiBLAS

- Initial idea: Summer 2013 after struggling with the linking issue.
- First release: December 2013 (BLAS and CBLAS only)
- Presented at GAMM '14, PMAA '14, OctConf '15.
- 2015-2017 code rewrite and use of code-generators.
- Current Public Version: 2.0 (April 2017)
- Provides interfaces for BLAS, CBLAS, and LAPACK.

How does it work?
General Approach

Long Story Short

We employ a plugin-like framework on top of the POSIX features for dynamic loading of shared libraries at runtime.

How does it work?
General Approach

Long Story Short

We employ a plugin-like framework on top of the POSIX features for dynamic loading of shared libraries at runtime.

POSIX.1 2001 `dl*`-family

`dlopen` add a shared library and its dynamic dependencies to the current address space.

`dlsym` search for symbols in the current address space beginning in the handle retrieved by `dlopen`.

`dlclose` close a previously opened shared library if no other references to the library exist.

`dlerror` provide human readable error messages.

How does it work?
General Approach

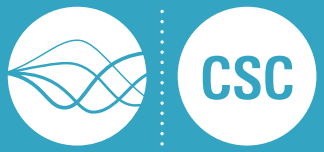
`dlopen` based issues to solve

1. `dlopen` only integrates selected parts of the library:
   Each required BLAS call needs to be initialized separately.
2. Dynamically (runtime) loaded symbols can not be resolved while linking a program.
3. `dlopen` only loads a single file:
   Multi-file implementations require additional treatment.

How does it work?
Initialization

`__attribute__((constructor))`

- automatically executed before the program starts.
- replaces deprecated `_init()`.
- Here used to read configuration and explicitly resolve all BLAS-routines to make sure they get loaded by `dlopen` as an initialization stage.



How does it work?
Initialization

`__attribute__((constructor))`

- automatically executed before the program starts.
- replaces deprecated `_init()`.
- Here used to read configuration and explicitly resolve all BLAS-routines to make sure they get loaded by `dlopen` as an initialization stage.

`__attribute__((destructor))`

- automatically executed after the main program exits.
- replaces deprecated `_fini()`.
- Here used to cleanly close the loaded shared library and potentially print profiling data.

How does it work?

Wrapper Functions

Goal

Provide a 100% Netlib-BLAS compatible API and ABI for use in user applications.

Goal

Provide a 100% Netlib-BLAS compatible API and ABI for use in user applications.

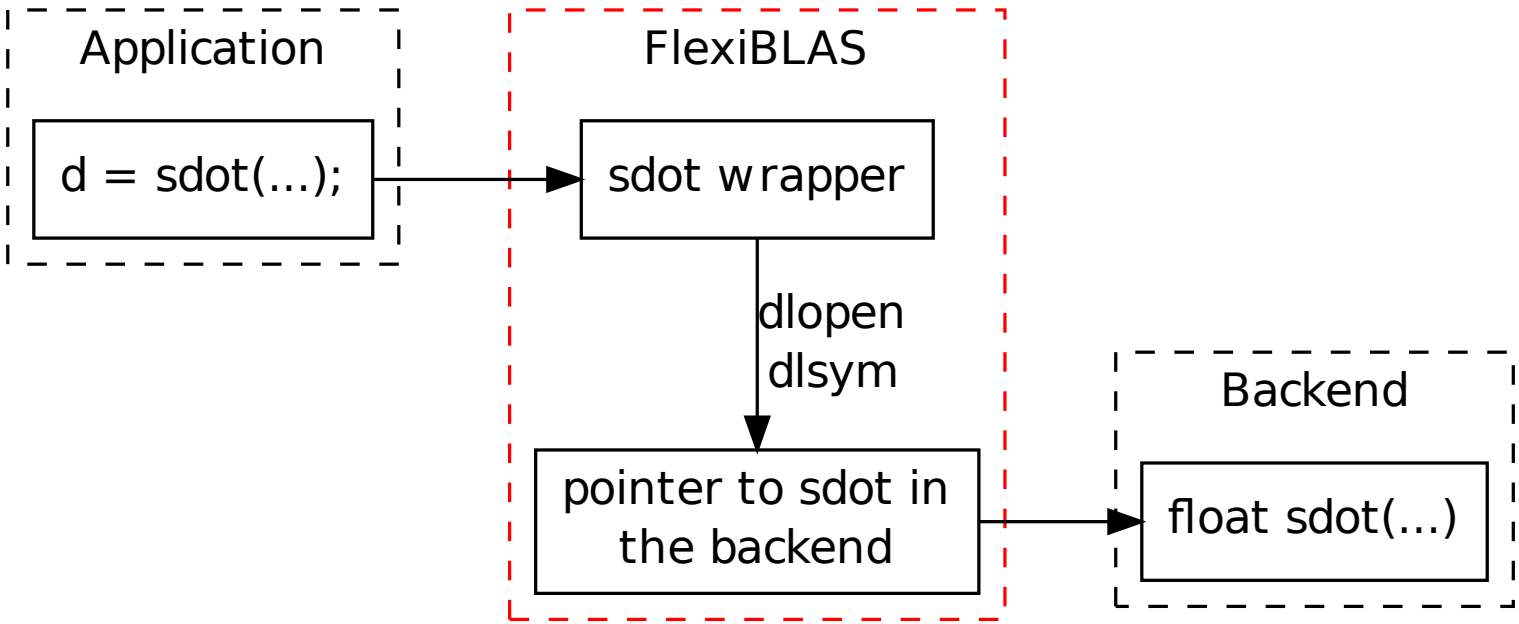


Figure: Calling `sdot` from an application via FlexiBLAS.

How does it work?

Wrapper Functions

Python based code-gen

- NumPy's `f2py` module allows to parse f77/f90 function headers.
- Extracted function headers are translated into Fortran-ABI compatible C functions containing the wrapper.

How does it work?
Wrapper Functions

Python based code-gen

- NumPy's `f2py` module allows to parse f77/f90 function headers.
- Extracted function headers are translated into Fortran-ABI compatible C functions containing the wrapper.

From

```
SUBROUTINE DAXPY(N, ALPHA, X, INCX, Y, INCY)
```

we obtain

```
void daxpy_(Int *N, double *ALPHA, double *X,
 Int *INCX,double *Y, Int * INCY) {
  ...
  fncall_daxpy(N,ALPHA,X,INCX,Y,INCY);
  ... }
```

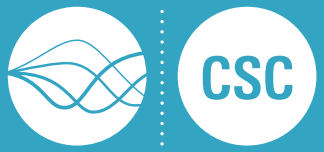
How does it work?
Overloading Functions

All BLAS routines can be overloaded to:

- build a deep profiling framework, (work in progress)
- dynamically offload them to accelerators, (work in progress)
- introduce faulty behavior for debugging purpose,
- original BLAS implementation is callable by a separate pointer.

Example - `DASUM` with perturbed output

```
double hook_dasum(Int *N, double *X, Int *INCX) {
 double res = fncall_real_dasum(N,X,INCX);
 return res + ((*N)*eps());
}
```



How does it work?

Profiling Framework

Functionality

- Collects all arguments, excluding arrays, of all BLAS calls.
- Measures the runtime of each BLAS call.
- Detects multi-threaded execution of the BLAS library.
- Stores all results in an SQLITE database.

Implementation

- Using NumPy's f2py again.
- Include a hook function for each BLAS call collecting the information.
- Uses the `constructor` and `destructor` routine for pre/post-processing.

How does it work?

Profiling Framework

Functionality

- Collects all arguments, excluding arrays, of all BLAS calls.
- Measures the runtime of each BLAS call.
- Detec
- Stores

Why?:

- Replay all BLAS calls to find errors in BLAS implementations.
  (e.g. OpenBLAS bugs #1332, #237, #1191)
- See how good "blackbox" codes utilizes the BLAS library.
- ...

Implement

- Using
- Includ formation.
- Uses the `constructor` and `destructor` routine for pre/post-processing.

How does it work?
Multi-file BLAS treatment

Remaining Question

How do we treat BLAS libraries consisting of multiple files (e.g. MKL and some versions of ATLAS), when the `dl*`-family can only use single file shared object libraries?

How does it work?
Multi-file BLAS treatment

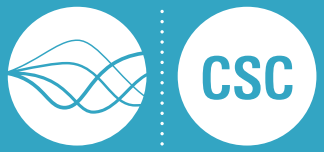
Remaining Question

How do we treat BLAS libraries consisting of multiple files (e.g. MKL and some versions of ATLAS), when the `dl*`-family can only use single file shared object libraries?

Simple trick

Place an additional surrogate library between FlexiBLAS and, e.g., MKL that references all necessary symbols in MKL and behaves like a netlib-BLAS interface from the view of the dynamic linker.

Intel MKL provides a set of `Makefiles` to create such dummy libraries containing arbitrary BLAS symbols.



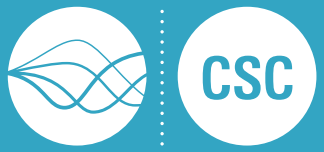
How does it work?

What else is implemented in version 2.0?

- Wrappers around some additional functions from OpenBLAS,
- Wrappers for all routines of LAPACK 3.6.1,
- Command line tool for easy management,
- API to change the BLAS backend at runtime,
- GNU Octave interface for the API.
- Library to manage the configuration files,
- Packaging scripts for Ubuntu/Debian.

Planed for version 3.0:

- Fine grained profiling.
- Increased LAPACK compatibility.



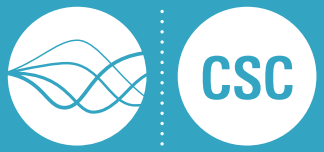
How is it used?

We provide a tool that closely follows Gentoo's `eselect` syntax.
To check for backends, do

```
flexiblas list
```

To select the active backend, use

```
flexiblas default BLAS_BACKEND_NAME
```



How is it used?

We provide a tool that closely follows Gentoo's `eselect` syntax.
To check for backends, do

```
flexiblas list
```

To select the active backend, use

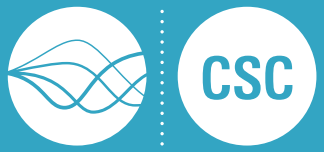
```
flexiblas default BLAS_BACKEND_NAME
```

Alternatively we use an environment variable as in:

```
export FLEXIBLAS=/usr/lib/libopenblas.so
```

or

```
export FLEXIBLAS=ATLAS
```



How is it used?

We provide a tool that closely follows Gentoo's `eselect` syntax.
To check for backends, do

```
flexiblas list
```

To select the active backend, use

```
flexiblas default BLAS_BACKEND_NAME
```

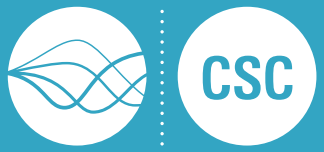
Alternatively we use an environment variable as in:

```
export FLEXIBLAS=/usr/lib/libopenblas.so
```

or

```
export FLEXIBLAS=ATLAS
```

Both rely on configuration files generated automatically in `/etc/flexiblasrc` and `~/.flexiblasrc`



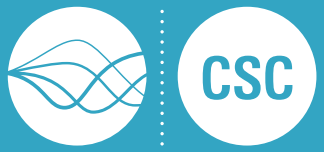
How is it used?

New BLAS libraries can be added by:

```
flexiblas add BLASNAME sharedlibrary.so
```

or other runtime properties, like verbosity or easy profiling, can be set:

```
flexiblas set PROPERTY VALUE
```



Future Plans?

- Tests with the LLVM/CLang/FLang Compiler Suite,
- Tests with IBM XLC/XLF on ppc64le,
- Keep track of the BLAS enhancements,
- Ongoing update of LAPACK,
- Extend the profiling framework,
- Get the automatic offloading ready, ↗ AutoBLAS
- **Get into the distributions!!!** (at the moment only Arch/AUR).

Details

M. Köhler and J. Saak, *FlexiBLAS - A flexible BLAS library with runtime exchangeable backends*, Tech. Rep. 284, LAPACK Working Note, Jan. 2014.

Future Plans?

- Tests with the LLVM/CLang/FLang Compiler Suite,
- Tests with IBM XLC/XLF on ppc64le,
- Keep track of the BLAS enhancements,
- Ongoing update of LAPACK,
- Extend the profiling framework,
- Get the automatic offload feature of AutoBLAS
- **Get into the distribution** (at the moment only Arch/AUR).

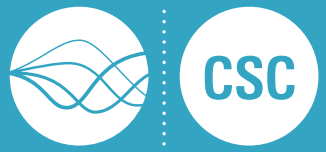
Thank you very much for your attention!

for the software package visit:

`http://www.mpi-magdeburg.mpg.de/projects/flexiblas`

Details

M. Köhler and J. Saak, *FlexiBLAS - A flexible BLAS library with runtime exchangeable backends*, Tech. Rep. 284, LAPACK Working Note, Jan. 2014.



Wishes? Ideas? What do you need?