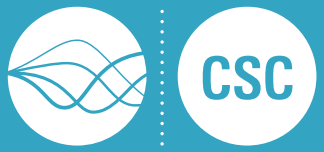
Profiling and Inspecting Linear Algebra Applications using FlexiBLAS

Martin Köhler

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

March 21, 2018

89th GAMM Annual Meeting

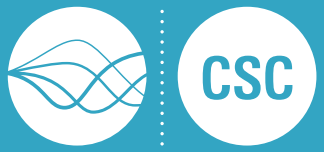


What is BLAS?

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?

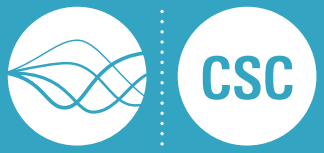
Basic Linear Algebra Subprograms (BLAS)

"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."[6]

Used in many software packages:

- MATLAB, GNU Octave
- NumPy, SciPy
- Julia
- FEM Packages like FEniCS, DEAL.II, . . .
- Libraries like PetSC, Trillinos, Eigen, SuiteSparse, . . .

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



What is BLAS?

Some important BLAS implementations:

- 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/` (automatic compile-time tuning)
- BLIS (BLAS-like Library Instantiation Software Framework):
  `https://github.com/flame/blis` (alternative approach to BLAS)
- Intel® Math kernel library (MKL):
  `http://software.intel.com/en-us/intel-mkl/` (fastest on Intel CPUs)
- Apple Accelerate, IBM ESSL, ...

What is BLAS?

Some important BLAS implementations:

- 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/` (automatic compile-time tuning)
- BLIS (BLAS-like Library Instantiation Software Framework): `https://github.com/flame/blis` (alternative approach to BLAS)
- Intel® Math kernel library (MKL): `http://software.intel.com/en-us/intel-mkl/` (fastest on Intel CPUs)
- Apple Accelerate, IBM ESSL, ...

FlexiBLAS

- Lightweight wrapper library around all BLAS and LAPACK subroutines using a plugin framework.
- Ability to overload all BLAS and LAPACK subroutines.
- Developed since 2013.

Why do we develop FlexiBLAS?

Main Reason: Linker Problems

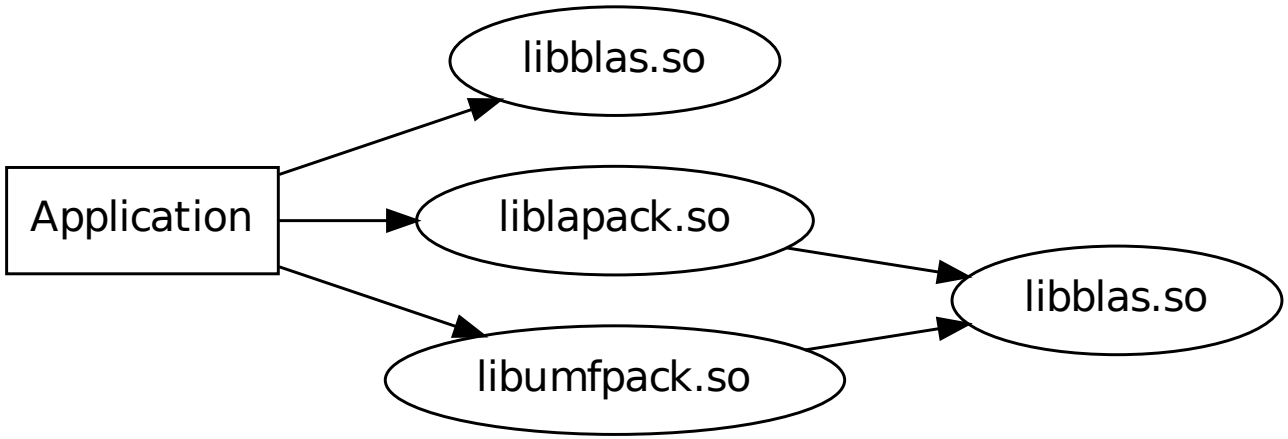


Figure: A sample application using BLAS

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

Why do we develop FlexiBLAS?

Main Reason: Linker Problems

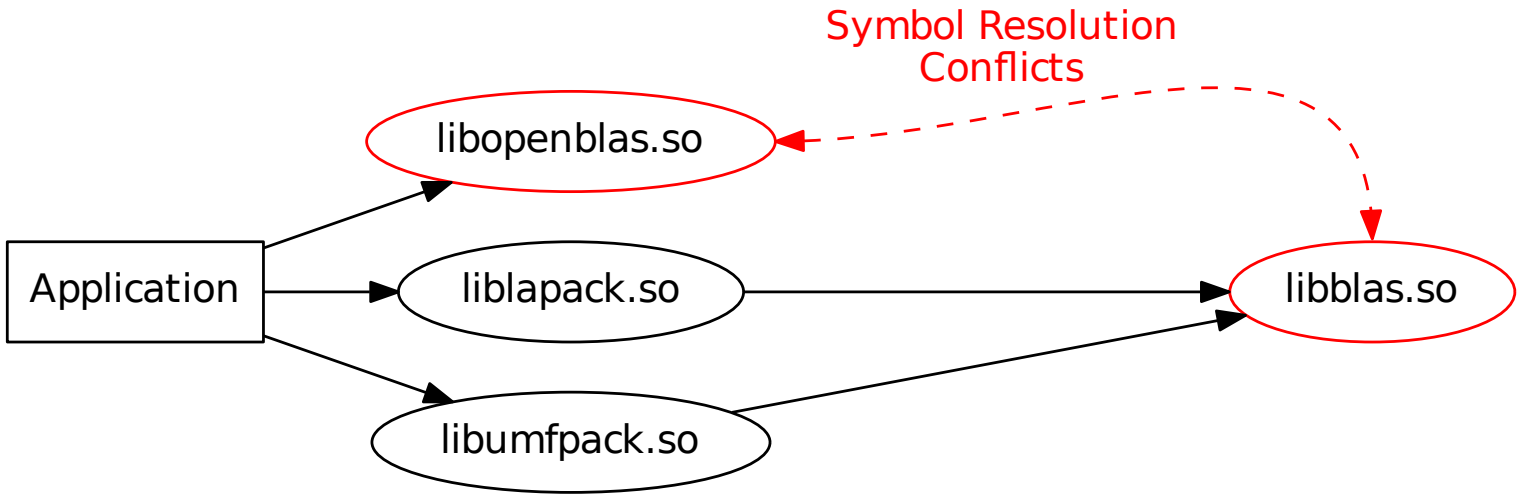


Figure: A sample application using BLAS

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

Why do we develop FlexiBLAS?

Main Reason: Linker Problems

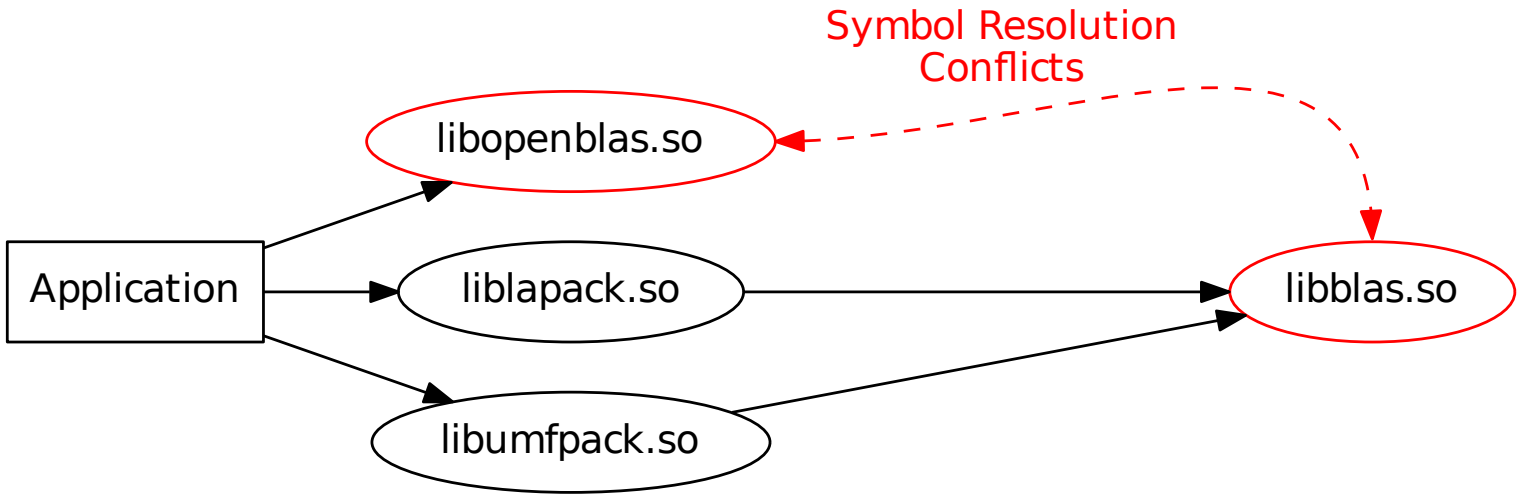
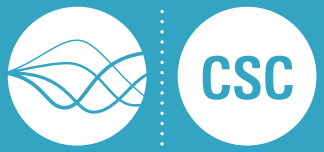


Figure: A sample application using BLAS

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

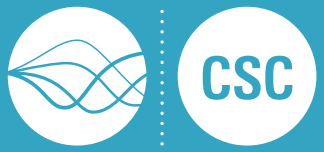
With FlexiBLAS one switches the BLAS library without relinking and producing such conflicts.



Why do we develop FlexiBLAS?

Further Reasons

- **Compatibility:** Different interface styles of GNU Fortran, Intel Fortran,...
- **Vendor-added Routines:** `xAXPBY`, `xOMATCOPY`, ...
- **Bad Linking Behavior:** e.g. longish linker setups for MKL or ATLAS.
- **Time consuming rebuilding:** Switching BLAS and LAPACK may require rebuilding a whole project.
- **Super User Privileges:** Most solutions provided by OS vendors require administrator rights for this.

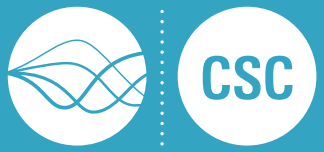


Why do we develop FlexiBLAS?

Further Reasons

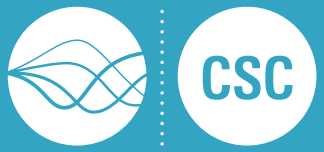
- **Compatibility:** Different interface styles of GNU Fortran, Intel Fortran,...
- **Vendor-added Routines:** `xAXPBY`, `xOMATCOPY`, ...
- **Bad Linking Behavior:** e.g. longish linker setups for MKL or ATLAS.
- **Time consuming rebuilding:** Switching BLAS and LAPACK may require rebuilding a whole project.
- **Super User Privileges:** Most solutions provided by OS vendors require administrator rights for this.

FlexiBLAS provides easy to use solutions for these problems.



Profiling

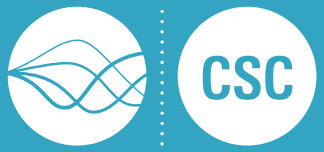
- Profiling usually requires additional compiler settings.
- Profiling data requires additional (sometimes confusing) tools for evaluation.
- Profiling often induce considerable overhead influencing the runtime behavior of the profiled application.
- **Hardly possible if only precompiled binaries are available.**



Profiling

- Profiling usually requires additional compiler settings.
- Profiling data requires additional (sometimes confusing) tools for evaluation.
- Profiling often induce considerable overhead influencing the runtime behavior of the profiled application.
- **Hardly possible if only precompiled binaries are available.**

However, what is happening behind high-level interfaces like Octave, SciPy, . . . ?



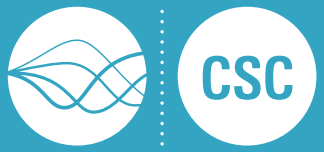
Profiling

- Profiling usually requires additional compiler settings.
- Profiling data requires additional (sometimes confusing) tools for evaluation.
- Profiling often induce considerable overhead influencing the runtime behavior of the profiled application.
- **Hardly possible if only precompiled binaries are available.**

However, what is happening behind high-level interfaces like Octave, SciPy, . . . ?

Our Focus

To get an idea of what is going on in the application, we need to collect some basic statistics.



Profiling

- Profiling usually requires additional compiler settings.
- Profiling data requires additional (sometimes confusing) tools for evaluation.
- Profiling often induce considerable overhead influencing the runtime behavior of the profiled application.
- **Hardly possible if only precompiled binaries are available.**

However, what is happening behind high-level interfaces like Octave, SciPy, . . . ?

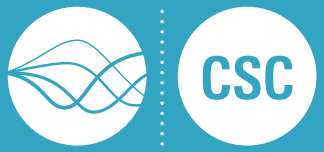
Our Focus

To get an idea of what is going on in the application, we need to collect some basic statistics.

Basic profiling should:

- **count the number of calls** for each BLAS subroutine
- and **measure the runtime** taken by the BLAS library

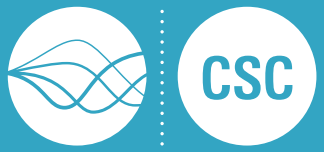
without recompiling/patching/modifying the application.



Inspecting

Similar to profiling but:

- works **for each function/subroutine call** individually
- and gathers runtime, function/subroutine arguments, threads-ids, ...



Inspecting

Similar to profiling but:

- works **for each function/subroutine call** individually
- and gathers runtime, function/subroutine arguments, threads-ids, ...

Inspecting normally requires:

- code compiled with debug information or annotations,
- tools for dynamic insertion of code into binaries, (e.g.: Dyninst, Score-P)
- OS support for various tracing tools, (e.g.: DTrace, FTrace)
- modified (wrapper) libraries, (e.g.: eztrace)
- or ugly tricks using `LD_PRELOAD`. (e.g.: eztrace)

Inspecting

Similar to profiling but:

- works **for each function/subroutine call** individually
- and gathers runtime, function/subroutine arguments, threads-ids, ...

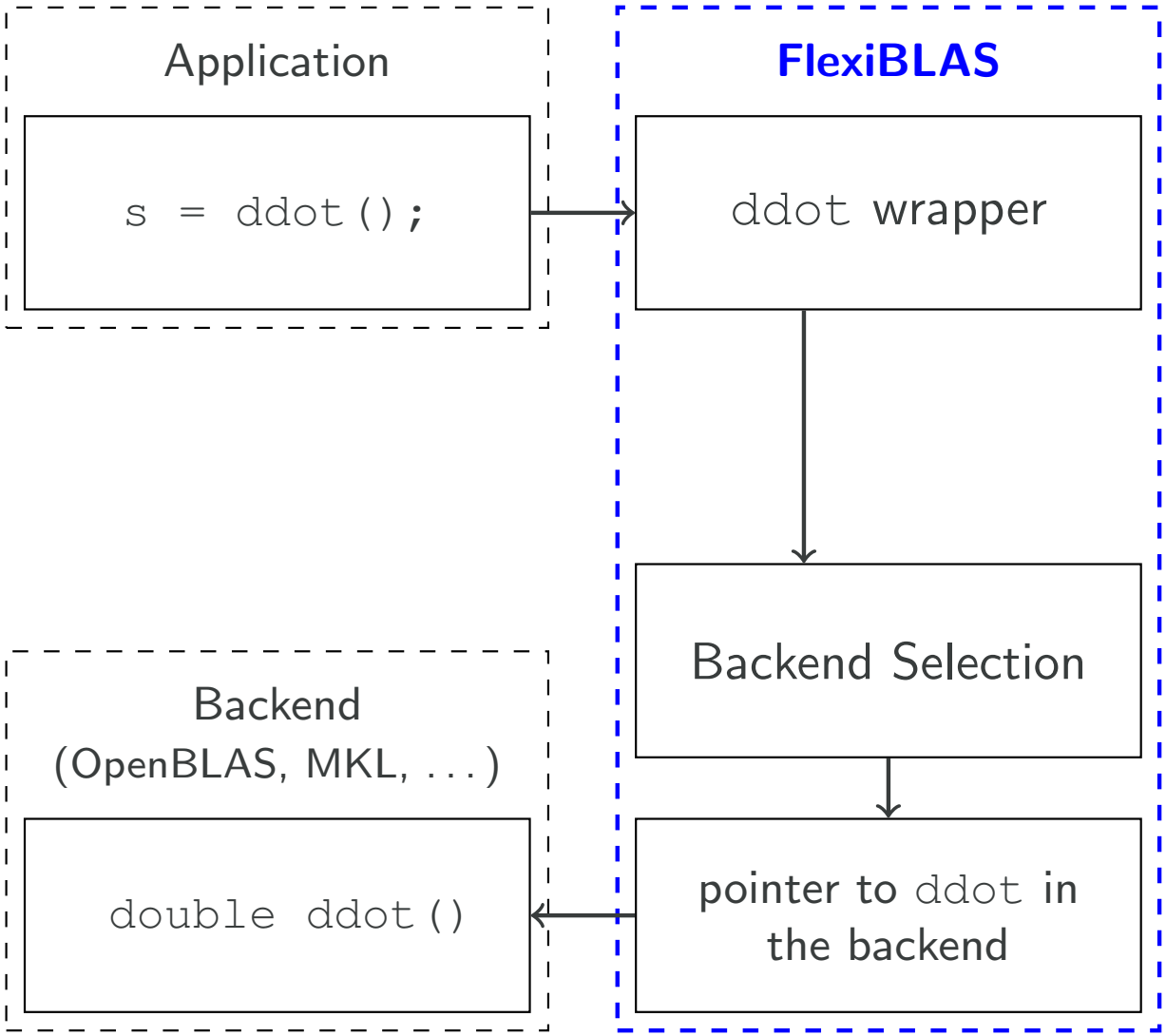
Inspecting normally requires:

- code compiled with debug information or annotations,
- tools for dynamic insertion of code into binaries, (e.g.: Dyninst, Score-P)
- OS support for various tracing tools, (e.g.: DTrace, FTrace)
- modified (wrapper) libraries, (e.g.: eztrace)
- or ugly tricks using `LD_PRELOAD`. (e.g.: eztrace)

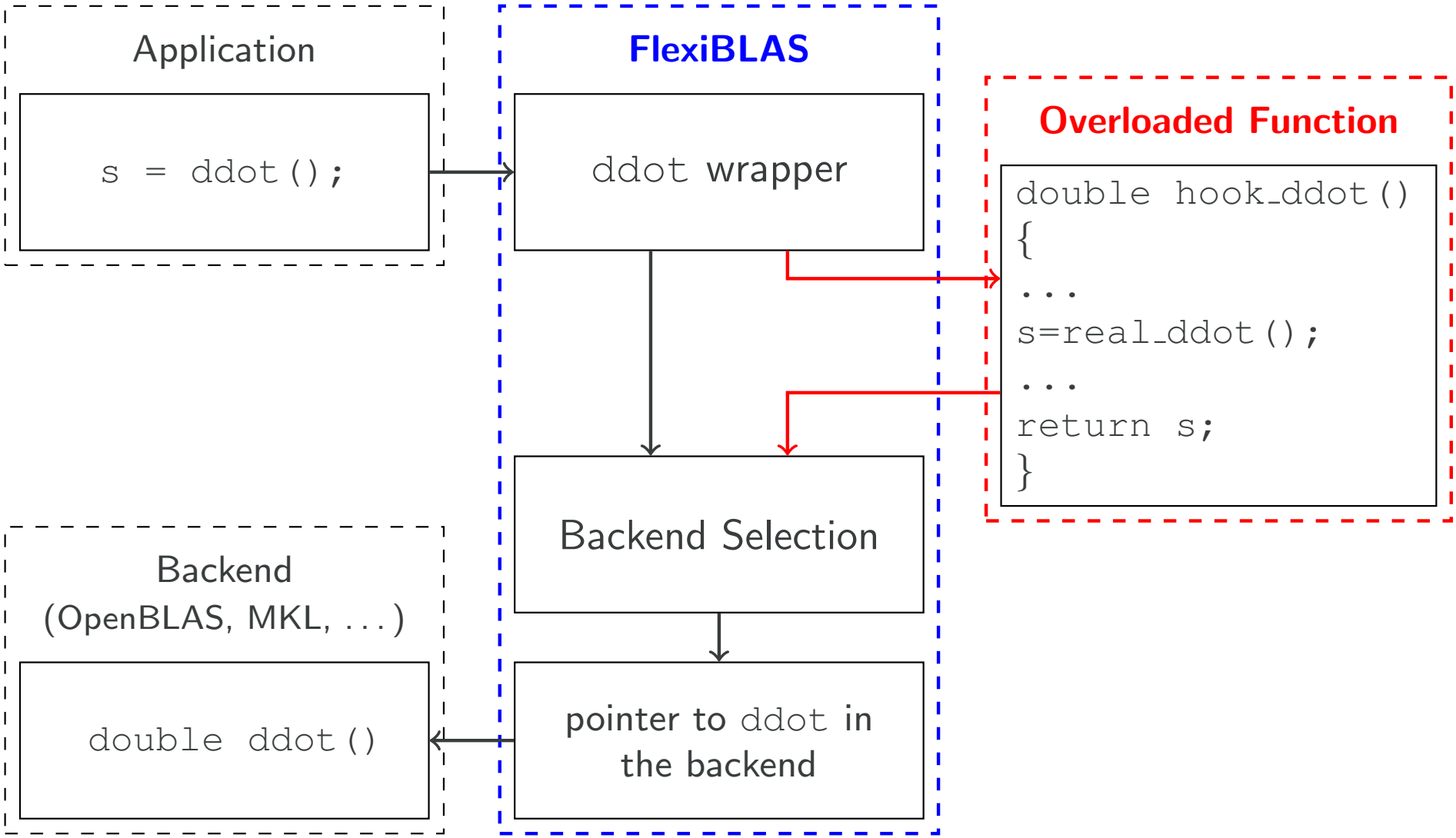
Our Goal

Collect the meta information of all BLAS calls including scalar function arguments, starting and finishing times, and the id of the executing thread.

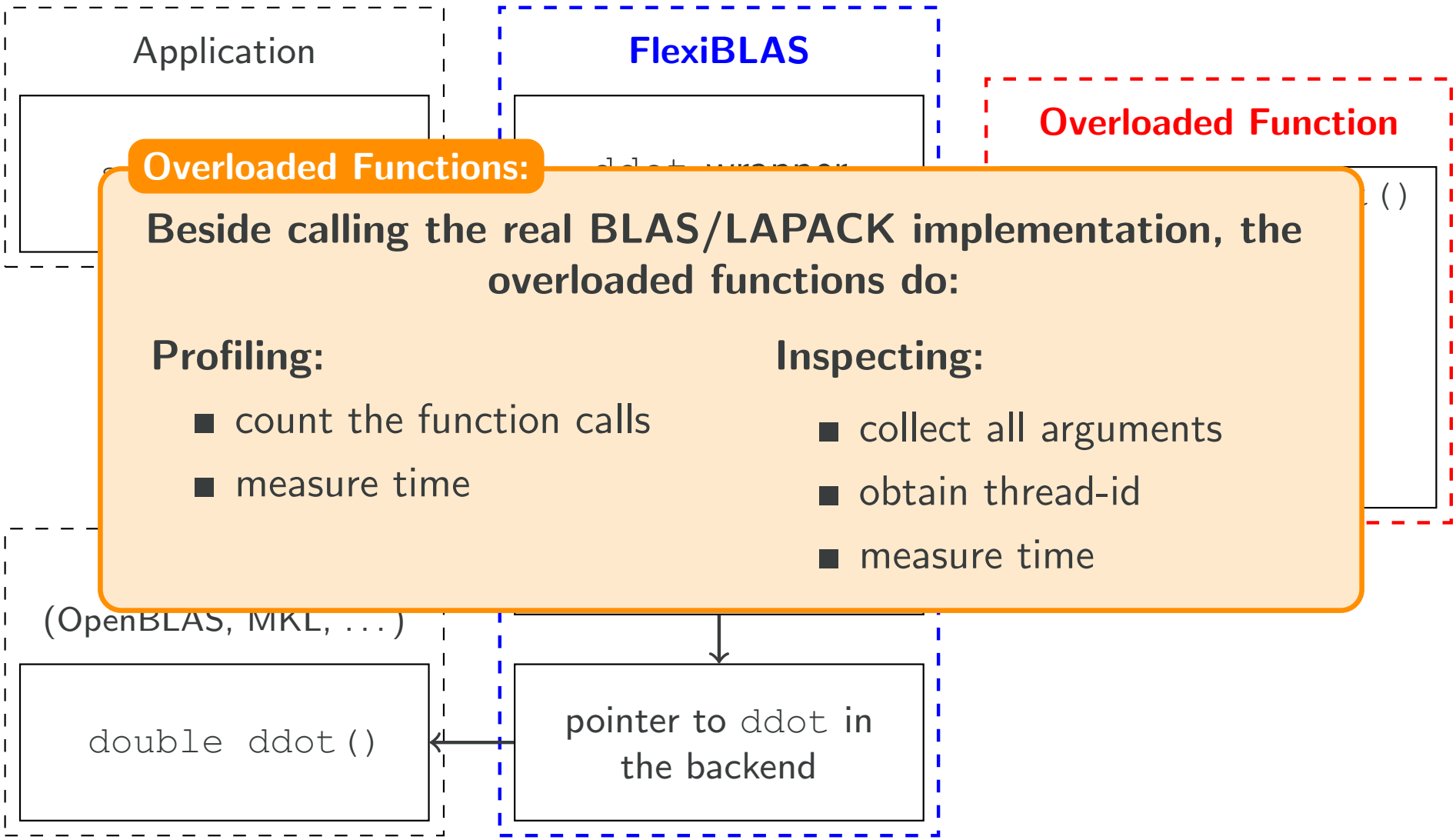
How is it Implemented?

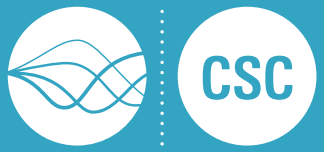


How is it Implemented?



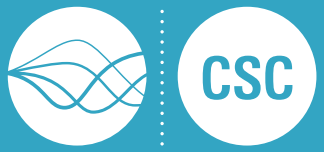
How is it Implemented?





How is it Implemented?

We have $\approx$ 140 BLAS and more than 1 500 LAPACK subroutines.

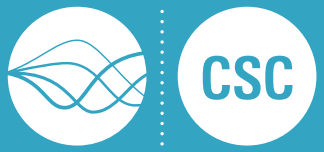


How is it Implemented?

We have ≈ 140 BLAS and more than 1 500 LAPACK subroutines.

Python based code-gen

- NumPy's `f2py` module allows f77/f90 function headers to be parsed:
  - Extracts all subroutine/function headers.
  - Provides information about scalar and array arguments.
- FlexiBLAS uses the Python template engines to create Fortran-ABI compatible C functions for all extracted subroutines/functions.



How is it Implemented?

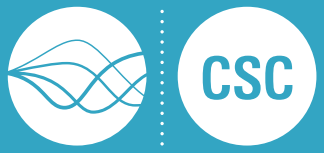
We have ≈ 140 BLAS and more than 1 500 LAPACK subroutines.

Python based code-gen

- NumPy's `f2py` module allows f77/f90 function headers to be parsed:
  - Extracts all subroutine/function headers.
  - Provides information about scalar and array arguments.
- FlexiBLAS uses the Python template engines to create Fortran-ABI compatible C functions for all extracted subroutines/functions.

Storing Inspection Data

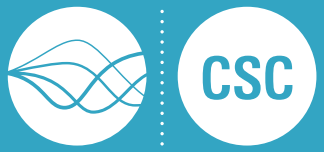
- Everything is stored in a fixed size buffer. (50 000 entries by default)
- The buffer is secured using atomic variables or mutexes.
- If the buffer is full or the program ends, the buffer is flushed to an SQLite database.
  → Fast and structured access to the collected data.



FlexiBLAS Features

Profile Viewer

Extract basic profiling data to a human-readable format or convert it into an easily processable format.



FlexiBLAS Features

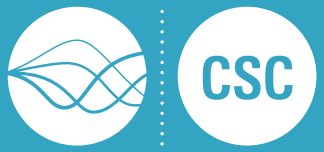
Profile Viewer

Extract basic profiling data to a human-readable format or convert it into an easily processable format.

Function Call Replay

Execute each BLAS/LAPACK call again with random data

- to compare against Netlib BLAS for correctness
- or to search the fastest BLAS library for this operation.



FlexiBLAS Features

Profile Viewer

Extract basic profiling data to a human-readable format or convert it into an easily processable format.

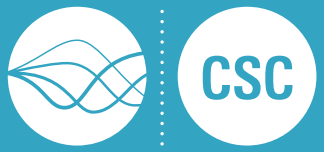
Function Call Replay

Execute each BLAS/LAPACK call again with random data

- to compare against Netlib BLAS for correctness
- or to search the fastest BLAS library for this operation.

Trace Generation

Create timelines from all performed BLAS/LAPACK calls including their thread affinity to visualize parallel workflows.



FlexiBLAS Features

Profile Viewer

Extract basic profiling data to a human-readable format or convert it into an easily processable format.

Function Call Replay

Execute each BLAS/LAPACK call again with random data

- to compare against Netlib BLAS for correctness
- or to search the fastest BLAS library for this operation.

Trace Generation

Create timelines from all performed BLAS/LAPACK calls including their thread affinity to visualize parallel workflows.

Alternative Usage of Overloading Capabilities – Automatic Offload

The collected data can help determine if there is a benefit to offloading certain operations to accelerator devices (GPUs, Xeon Phis, FPGAs). FlexiBLAS allows this to be done automatically per BLAS/LAPACK function.

Example Usage
BLAS and GNU Octave 4.2.1

How does GNU Octave use the BLAS library?

```
function [x] = conjgrad(A, b, x)
  r = b - A * x;
  p = r;
  rsold = r' * r;
  for i = 1:length(b)
    Ap = A * p;
    alpha = rsold / (p' * Ap);
    x = x + alpha * p;
    r = r - alpha * Ap;
    rsnew = r' * r;
    if sqrt(rsnew) < 1e-10
       break;
    end;
    p = r + (rsnew / rsold) * p;
    rsold = rsnew;
 end
end

A = full(sprandsym(1000,1.0));
b = A*ones(1000,1);
x = conjgrad(A,b,zeros(1000,1));
norm(A*x-b)/norm(b)
```

Example Usage
BLAS and GNU Octave 4.2.1

How does GNU Octave use the BLAS library?

```
function [x] = conjgrad(A, b, x)
  r = b - A * x;
  p = r;
  rsold = r' * r;
  for i = 1:length(b)
    Ap = A * p;
    alpha = rsold / (p' * Ap);
    x = x + alpha * p;
    r = r - alpha * Ap;
    rsnew = r' * r;
    if sqrt(rsnew) < 1e-10
       break;
    end;
    p = r + (rsnew / rsold) * p;
    rsold = rsnew;
 end
end

A = full(sprandsym(1000,1.0));
b = A*ones(1000,1);
x = conjgrad(A,b,zeros(1000,1));
norm(A*x-b)/norm(b)
```

Profiling:

Subroutine	# Calls	acc. Time
ddot	1000	1.11e-03s
dgemv	1003	4.04e-01s
dsyrk	**1001**	**5.61e-03s**
dlamch	5	2.69e-05s

Observations

- Vector addition/scaling/norms not mapped to BLAS.
- Where does the symmetric rank-k update come from?

How does GNU Octave use the BLAS library?

```
function [x] = conjgrad(A, b, x)
  r = b - A * x;
  p = r;
  rsold = r' * r;
  for i = 1:length(b)
    Ap = A * p;
    alpha = rsold / (p' * Ap);
    x = x + alpha * p;
    r = r - alpha * Ap;
    rsnew = r' * r;
    if sqrt(rsnew) < 1e-10
       break;
    end;
    p = r + (rsnew / rsold) * p;
    rsold = rsnew;
 end
end

A = full(sprandsym(1000,1.0));
b = A*ones(1000,1);
x = conjgrad(A,b,zeros(1000,1));
norm(A*x-b)/norm(b)
```

Inspecting:

`DSYRK` computes

$$C := \alpha A \cdot A^T + \beta C \quad \text{if } \texttt{trans='N'}$$

or

$$C := \alpha A^T A + \beta C, \quad \text{if } \texttt{trans='T'}$$

with $A \in \mathbb{R}^{n \times k}$ or $A \in \mathbb{R}^{k \times n}$ and $C \in \mathbb{R}^{n \times n}$.

All 1001 DSYRK calls use:

- `trans = 'T'`
- $n = 1$, $k = 1000$,
- $\alpha = 1.0$, and $\beta = 0$

How does GNU Octave use the BLAS library?

```
function [x] = conjgrad(A, b, x)
  r = b - A * x;
  p = r;
  rsold = r' * r;
  for i = 1:length(b)
    Ap = A * p;
    alpha = rsold / (p' * Ap);
    x = x + alpha * p;
    r = r - alpha * Ap;
    rsnew = r' * r;
    if sqrt(rsnew) < 1e-10
       break;
    end;
    p = r + (rsnew / rsold) * p;
    rsold = rsnew;
 end
end

A = full(sprandsym(1000,1.0));
b = A*ones(1000,1);
x = conjgrad(A,b,zeros(1000,1));
norm(A*x-b)/norm(b)
```

Inspecting:

`DSYRK` computes

$$C := \alpha A \cdot A^T + \beta C \quad \text{if } \texttt{trans='N'}$$

or

$$C := \alpha A^T A + \beta C, \quad \text{if } \texttt{trans='T'}$$

with $A \in \mathbb{R}^{n \times k}$ or $A \in \mathbb{R}^{k \times n}$ and $C \in \mathbb{R}^{n \times n}$.

All 1001 `DSYRK` calls use:

- `trans = 'T'`
- $n = 1$, $k = 1000$,
- $\alpha = 1.0$, and $\beta = 0$

→ **Misuse of `DSYRK` to compute the squared 2-norm of a vector.**

Example Usage

OpenBLAS – Issue #1332

Bug Description: **(OpenBLAS up to version 0.2.20)**

The `DTRMV` routine computes

$$x := \alpha T x$$

with $\alpha \in \mathbb{R}$, $x \in \mathbb{R}^n$, and $T \in \mathbb{R}^{n \times n}$ upper or lower triangular.

For example we had production code, where T was stored with leading dimension 64 and n increased from 1 to 64 during an iterative process:

- if $n > 16$, the result x gets perturbed,
- if $n > 32$, the result x is completely wrong.

Example Usage

OpenBLAS – Issue #1332

Bug Description: **(OpenBLAS up to version 0.2.20)**

The `DTRMV` routine computes

$$x := \alpha T x$$

with $\alpha \in \mathbb{R}$, $x \in \mathbb{R}^n$, and $T \in \mathbb{R}^{n \times n}$ upper or lower triangular.

For example we had production code, where T was stored with leading dimension 64 and n increased from 1 to 64 during an iterative process:

- if $n > 16$, the result x gets perturbed,
- if $n > 32$, the result x is completely wrong.

Detection through replaying and comparison to Netlib BLAS:

```
...
Cor.  RESULT: DTRMV(U,N,N, 16, A, 64, X, 1) MAXERR = 0.00D+00
Pert. RESULT: DTRMV(U,N,N, 17, A, 64, X, 1) MAXERR = 0.56D-13
...
Pert. RESULT: DTRMV(U,N,N, 32, A, 64, X, 1) MAXERR = 0.58D-10
Wrong RESULT: DTRMV(U,N,N, 33, A, 64, X, 1) MAXERR = 0.59D+06
...
```

Example Usage
OpenBLAS – Issue #1332

Bug Description: **(OpenBLAS up to version 0.2.20)**

The DTRMV routine computes

$$x := \alpha Tx$$

with $\alpha \in \mathbb{R}$, $x \in \mathbb{R}^n$ and $T \in \mathbb{R}^{n \times n}$ upper triangular.

For example, in the reduction code, where T was stored with leading dimension 64 and n increased from 1 to 64 during an iterative process:

- if $n > 16$ the result x gets perturbed
- if $n > 32$ the result x is completely wrong

Detection through replaying and comparison to Netlib BLAS:

```
...
Corr. RESULT: DTRMV(U,N,N, 16, A, 64, X, 1) MAXERR = 0.00D+00
Pert. RESULT: DTRMV(U,N,N, 17, A, 64, X, 1) MAXERR = 0.56D-13
...
Pert. RESULT: DTRMV(U,N,N, 32, A, 64, X, 1) MAXERR = 0.58D-10
Wrong RESULT: DTRMV(U,N,N, 33, A, 64, X, 1) MAXERR = 0.59D+06
...
```

Race-Condition: The error only appears if OpenBLAS uses multi-threading on highly optimized platforms.

First Workaround: Threading for `xTRMV` is deactivated.

Current Situation:

- Seems to exist more than 10 years.
- Longish discussion (more than 60 comments).
- Still not clear where the race condition comes from.
- Also affects another race condition on the OpenPOWER 8 platform.
- The `DAXPY` operation seems to be involved as well.

Example Usage
OpenBLAS – Issue #1332

Bug Description: **(OpenBLAS up to version 0.2.20)**

The DTRMV routine computes

$$x := \alpha Tx$$

with $\alpha \in \mathbb{R}^n$ and $T \in \mathbb{R}^{n \times n}$ upper triangular.

For example, in the reduction code, where T was stored with leading dimension 64 and n increased from 1 to 64 during an iterative process:

- if $n > 16$ the result x gets perturbed
- if $n > 32$ the result x is completely wrong

Race-Condition: The error only appears if OpenBLAS uses multi-threading on highly optimized platforms.

First Workaround: Threading for `xTRMV` is deactivated.

Current Situation:

- Seems to exist more than 10 years.
- Longish discussion (more than 60 comments).
- Still not clear where the race condition comes from.
- Also affects another race condition on the OpenPOWER 8 platform.
- The `DAXPY` operation seems to be involved as well.

FlexiBLAS helps to accelerate the testing of different OpenBLAS configurations.

Detection through replaying and comparison to Netlib BLAS:

```
...
Cor. RESULT: DTRMV(U,N,N, 16, A, 64, X, 1) MAXERR = 0.00D+00
Per. RESULT: DTRMV(U,N,N, 32, A, 64, X, 1) MAXERR = 0.58D-13
...
Per. RESULT: DTRMV(U,N,N, 32, A, 64, X, 1) MAXERR = 0.58D-13
Wrong RESULT: DTRMV(U,N,N, 33, A, 64, X, 1) MAXERR = 0.59D+06
...
```

TileQR [Buttari et. al. '09 – '15]

Reformulation of the Storage Compact QR decomposition parallelized using Direct-Acylic-Graph(DAG) on top of OpenMP 4.

```
do k=1, min(mb,nb)
  !$omp task depend(inout:a(k,k))
  call dgeqrt(a(k,k))
  !$omp end task
  do j=k+1,nb
    !$omp task depend(in:a(k,k)) depend(inout:a(k,j))
     call dgemqrt(a(k,k), a(k,j))
    !$omp end task
  end do
  do i=k+1, mb
    !$omp task depend(inout:a(k,k), a(i,k))
    call dtpqrt(a(k,k), a(i,k))
    !$omp end task
    do j=k+1, nb
      !$omp task depend(in:a(k,k),a(i,k)) &
      ! & depend(inout:a(k,j),a(i,j))
      call dtpmqrt(a(k,k), a(i,k), a(k,j), a(i,j))
      !$omp end task
    end do
  end do
end do
```

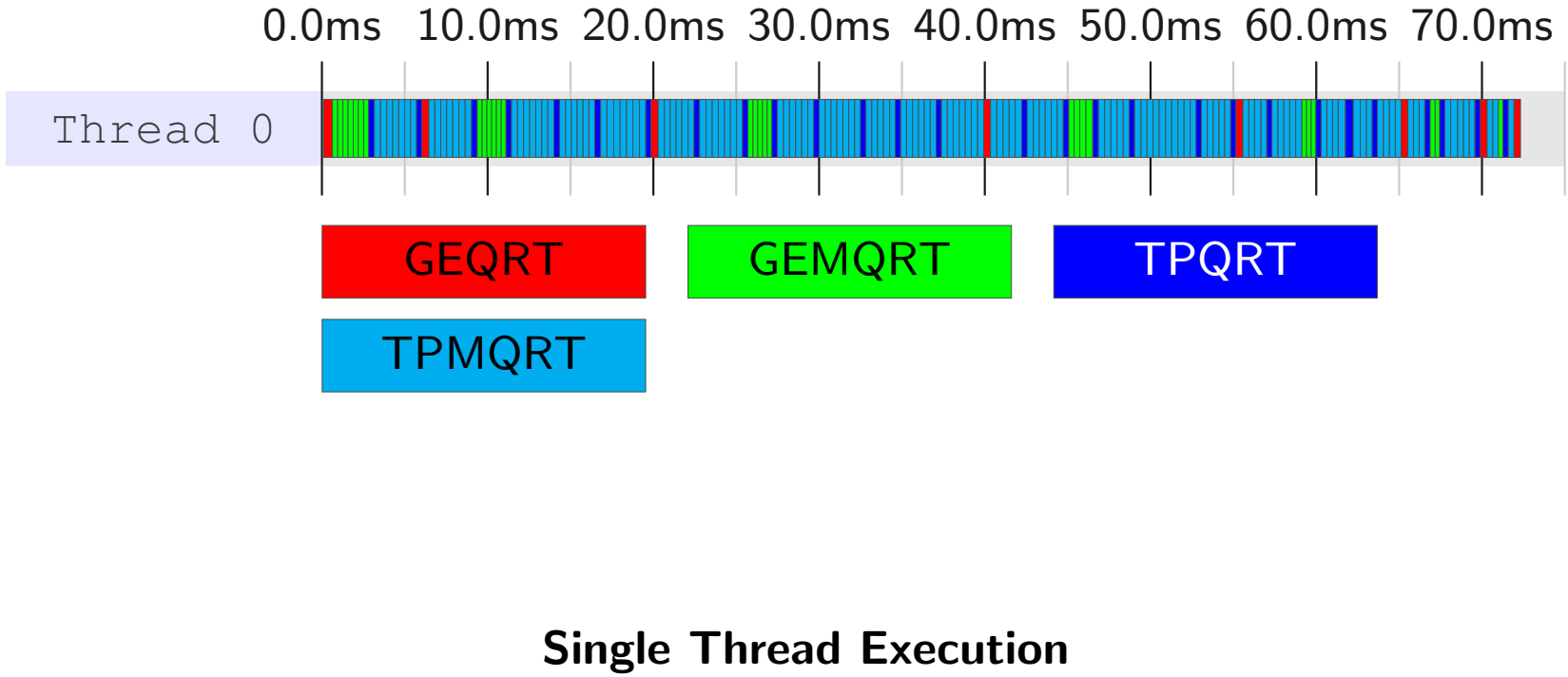
Scales well on Multi-Core CPU.

No threading support in BLAS required.

But how are the single LAPACK calls executed in parallel?

Visualization:

Use start-time, end-time, and thread-id from the SQLite database to visualize the program flow directly as TikZ/LaTeX:

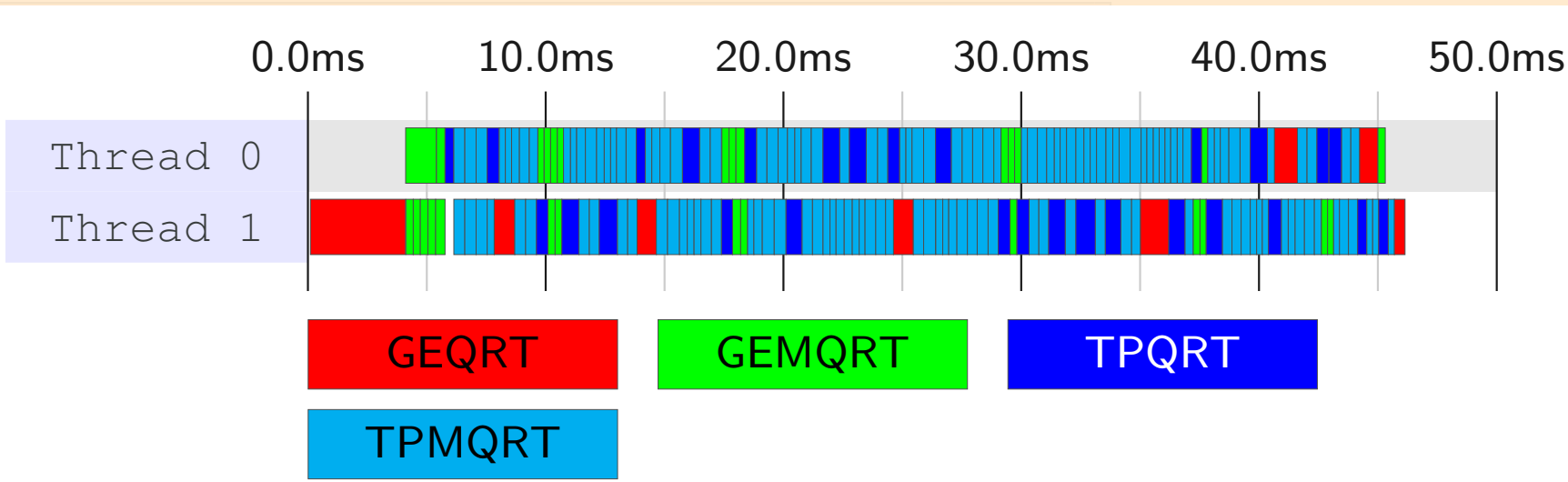


Single Thread Execution

```
    end do
  end do
end do
```

Visualization:

Use start-time, end-time, and thread-id from the SQLite database to visualize the program flow directly as TikZ/LaTeX:

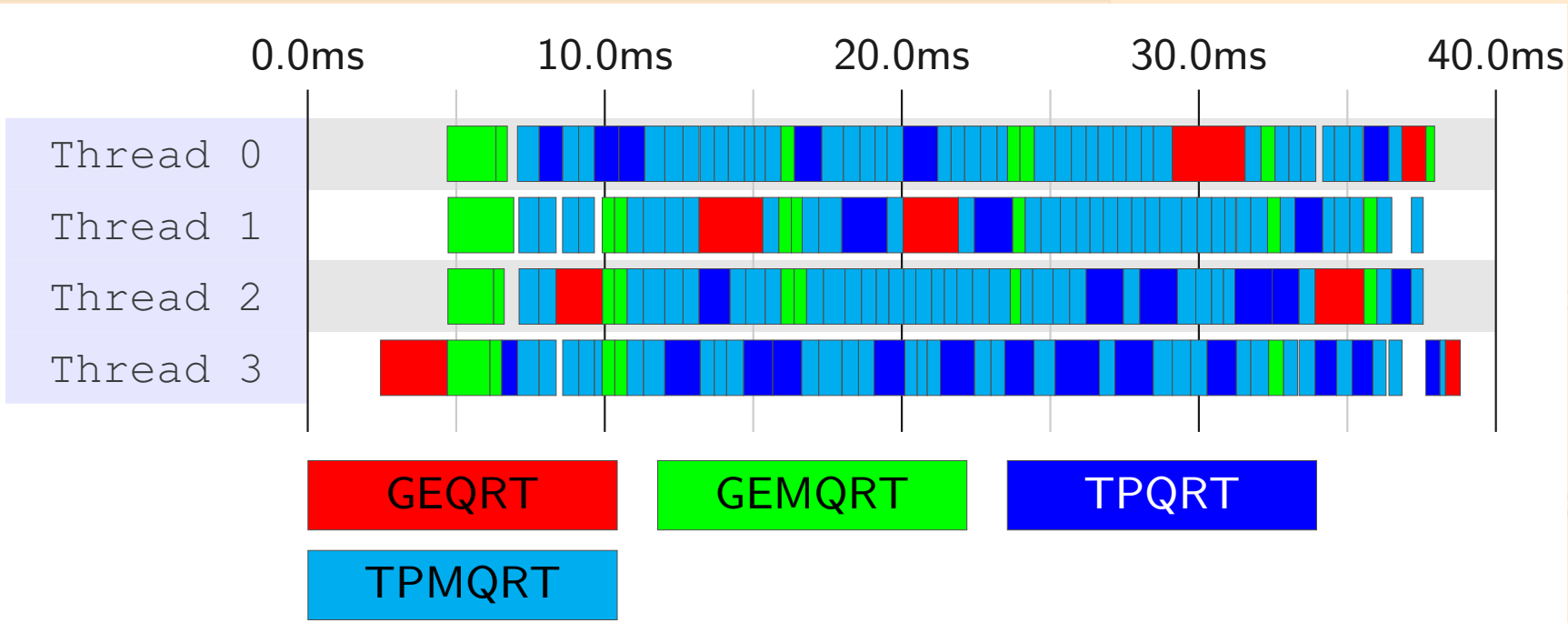


Two Thread Execution

```
    end do
  end do
end do
```

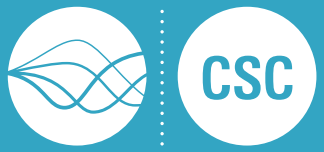
Visualization:

Use start-time, end-time, and thread-id from the SQLite database to visualize the program flow directly as TikZ/LaTeX:



Four Thread Execution

```
    end do
  end do
end do
```



Conclusions and Outlook

Conclusions

- Used FlexiBLAS's overloaded functions for debugging.
- Implemented a profiling/inspection using SQLite as a data store.
- Developed analysis tools to assist debugging and performance analysis.
- Found some interesting bugs in software packages (ours and others).

Outlook

- Develop more analysis tools and continue improving existing ones.
- Integrate automatic offloading as overloaded functions.

Details:

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

Conclusions and Outlook

Conclusions

- Used FlexiBLAS's overloaded functions for debugging

Thank you very much for your attention!

for the software package visit:

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

https://doi.org/10.5281/zenodo.569102

Ou

De

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