Which C compiler and BLAS/LAPACK library should I use – \textit{gretl}'s numerical efficiency in different configurations\textsuperscript{1}

Marcin Błażejowski

University WSB Merito in Toruń, Poland

Gretl Conference 2023

\textsuperscript{1}This research is a part of Oracle for Research grant (agreement No. v040119) entitled: "Which C compiler and which BLAS/LAPACK library? – gretl numerical efficiency in different configurations".
Workplan

1 Environment
   - Hardware
   - Libraries
   - Compilers

2 Compilation
   - CFLAGS in use
   - LAPACK_LIBS in use

3 Simulation scenarios
   - dgemm efficiency (pure BLAS Level 3)
   - LAPACK/BLAS efficiency
   - Real econometrics efficiency

4 Results
   - dgemm efficiency (pure BLAS Level 3)
   - LAPACK/BLAS efficiency
   - Real econometrics efficiency
   - Overall performance
6 core paravirtualized machines available in **Oracle Cloud** and run under **Canonical-Ubuntu-22.04-2023.02.15-0**:

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AMD</strong></td>
<td><strong>Intel</strong></td>
</tr>
<tr>
<td>EPYC 7J13 64-Core Processor</td>
<td>Xeon Gold 6354 CPU @ 3.00GHz</td>
</tr>
<tr>
<td>VM.Standard.E4.Flex</td>
<td>VM.Optimized3.Flex</td>
</tr>
</tbody>
</table>
Terminology

**BLAS** Basic Linear Algebra Subprograms

**LAPACK** Linear Algebra Package
Terminology

**BLAS** Basic Linear Algebra Subprograms
- originated as a Fortran library in 1979

**LAPACK** Linear Algebra Package
- originated as a Fortran library in 1992
Terminology

BLAS  **Basic Linear Algebra Subprograms**

- originated as a Fortran library in 1979
- stable release: 3.11.0 (November 11, 2022)

LAPACK  **Linear Algebra Package**

- originated as a Fortran library in 1992
- stable release: 3.11.0 (November 11, 2022)
Terminology

BLAS Basic Linear Algebra Subprograms

- originated as a Fortran library in 1979
- stable release: 3.11.0 (November 11, 2022)
- provides so-called: Level 1, Level 2 and Level 3 operations

LAPACK Linear Algebra Package

- originated as a Fortran library in 1992
- stable release: 3.11.0 (November 11, 2022)
**Terminology**

**BLAS**  *Basic Linear Algebra Subprograms*
- originated as a Fortran library in 1979
- stable release: 3.11.0 (November 11, 2022)
- provides so-called: Level 1, Level 2 and Level 3 operations

**LAPACK**  *Linear Algebra Package*
- originated as a Fortran library in 1992
- stable release: 3.11.0 (November 11, 2022)
- provides routines for solving systems of simultaneous linear equations, least-squares solutions, eigenvalue and singular value problems, matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur)
BLAS  **Basic Linear Algebra Subprograms**

- originated as a Fortran library in 1979
- stable release: 3.11.0 (November 11, 2022)
- provides so-called: Level 1, Level 2 and Level 3 operations

LAPACK  **Linear Algebra Package**

- originated as a Fortran library in 1992
- stable release: 3.11.0 (November 11, 2022)
- provides routines for solving systems of simultaneous linear equations, least-squares solutions, eigenvalue and singular value problems, matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur)
- relies on an underlying BLAS implementation
BLAS-only libraries

- **BLIS**: BLIS: based on GotoBLAS2 written by Kazushige Goto. Implements both: new and classic BLAS API, but no LAPACK API. Version: 0.9.0. Package used: libblis4-openmp 0.9.0-1.

BLAS-only libraries

- **BLIS:**
  - based on GotoBLAS2 written by Kazushige Gotô

- **AOCL-BLIS:**
BLAS-only libraries

- **BLIS**:  
  - based on GotoBLAS2 written by Kazushige Gotô  
  - implements both: new and classic BLAS API, but no LAPACK API

- **AOCL-BLIS**:
BLAS-only libraries

- **BLIS**:  
  - based on GotoBLAS2 written by Kazushige Gotô  
  - implements both: new and classic BLAS API, but no LAPACK API  
  - version: 0.9.0  
  - package used: `libblis4-openmp 0.9.0-1`

- **AOCL-BLIS**: 
BLAS-only libraries

- **BLIS**:  
  - based on GotoBLAS2 written by Kazushige Gotō  
  - implements both: new and classic BLAS API, but no LAPACK API  
  - version: 0.9.0  
  - package used: `libblis4-openmp 0.9.0-1`

- **AOCL-BLIS**:  
  - based on BLIS
BLAS-only libraries

- **BLIS:***
  - based on GotoBLAS2 written by Kazushige Gotō
  - implements both: new and classic BLAS API, but no LAPACK API
  - version: 0.9.0
  - package used: *libblis4-openmp 0.9.0-1*

- **AOCL-BLIS:**
  - based on BLIS
  - part of AMD Optimizing CPU Libraries (AOCL)
BLAS-only libraries

- **BLIS:**
  - based on GotoBLAS2 written by Kazushige Gotō
  - implements both: new and classic BLAS API, but no LAPACK API
  - version: 0.9.0
  - package used: libblis4-openmp 0.9.0-1

- **AOCL-BLIS:**
  - based on BLIS
  - part of AMD Optimizing CPU Libraries (AOCL)
  - version: 4.0.0 Build 20221031
  - package used: aocl-linux-aocc-4.0
LAPACK-only libraries

- **libFLAME:**

- **AOCL-libFLAME:**
LAPACK-only libraries

- **libFLAME:**
  - C-only implementation (no Fortran libraries dependency)

- **AOCL-libFLAME:**
LAPACK-only libraries

- **libFLAME:**
  - C-only implementation (no Fortran libraries dependency)
  - a complete dense linear algebra framework with LAPACK compatible API through `lapack2flame` layer

- **AOCL-libFLAME:**
LAPACK-only libraries

- **libFLAME:**
  - C-only implementation (no Fortran libraries dependency)
  - a complete dense linear algebra framework with LAPACK compatible API through `lapack2flame` layer
  - version: 5.2.0
  - package used: `libflame1 5.2.0-3ubuntu3`

- **AOCL-libFLAME:**
### LAPACK-only libraries

- **libFLAME:**
  - C-only implementation (no Fortran libraries dependency)
  - a complete dense linear algebra framework with LAPACK compatible API through `lapack2flame` layer
  - version: 5.2.0
  - package used: `libflame1 5.2.0-3ubuntu3`
  - abbreviation: `bflame` (as BLIS and libFLAME)

- **AOCL-libFLAME:**
LAPACK-only libraries

- **libFLAME:**
  - C-only implementation (no Fortran libraries dependency)
  - a complete dense linear algebra framework with LAPACK compatible API through `lapack2flame` layer
  - version: 5.2.0
  - package used: `libflame1 5.2.0-3ubuntu3`
  - abbreviation: `bflame` (as BLIS and libFLAME)

- **AOCL-libFLAME:**
  - based on libFLAME
LAPACK-only libraries

- **libFLAME:**
  - C-only implementation (no Fortran libraries dependency)
  - a complete dense linear algebra framework with LAPACK compatible API through `lapack2flame` layer
  - version: 5.2.0
  - package used: `libflame1 5.2.0-3ubuntu3`
  - abbreviation: `bflame` (as BLIS and libFLAME)

- **AOCL-libFLAME:**
  - based on libFLAME
  - part of AMD Optimizing CPU Libraries (AOCL)
LAPACK-only libraries

- **libFLAME**:  
  - C-only implementation (no Fortran libraries dependency)  
  - a complete dense linear algebra framework with LAPACK compatible API through `lapack2flame` layer  
  - version: 5.2.0  
  - package used: `libflame1 5.2.0-3ubuntu3`  
  - abbreviation: `bflame` (as BLIS and libFLAME)

- **AOCL-libFLAME**:  
  - based on libFLAME  
  - part of AMD Optimizing CPU Libraries (AOCL)  
  - version: 4.0.0 Build 20221031  
  - package used: `aocl-linux-aocc-4.0`
LAPACK-only libraries

- **libFLAME:**
  - C-only implementation (no Fortran libraries dependency)
  - a complete dense linear algebra framework with LAPACK compatible API through `lapack2flame` layer
  - version: 5.2.0
  - package used: `libflame1 5.2.0-3ubuntu3`
  - abbreviation: `bflame` (as BLIS and libFLAME)

- **AOCL-libFLAME:**
  - based on libFLAME
  - part of AMD Optimizing CPU Libraries (AOCL)
  - version: 4.0.0 Build 20221031
  - package used: `aocl-linux-aoccc-4.0`
  - abbreviation: `aocl` (as AOCL-BLIS with AOCL-libFLAME)
BLAS/LAPACK combo libraries

- **OpenBLAS:**

- **Intel oneAPI Math Kernel Library (oneMKL):**
BLAS/LAPACK combo libraries

- **OpenBLAS:**
  - based on GotoBLAS2 written by Kazushige Gotô

- **Intel oneAPI Math Kernel Library (oneMKL):**
BLAS/LAPACK combo libraries

- **OpenBLAS:**
  - based on GotoBLAS2 written by Kazushige Gotō
  - version: 0.3.20
  - package used: libopenblas0-openmp 0.3.20+ds-1

- **Intel oneAPI Math Kernel Library (oneMKL):**
BLAS/LAPACK combo libraries

- **OpenBLAS:**
  - based on GotoBLAS2 written by Kazushige Gotō
  - version: 0.3.20
  - package used: `libopenblas0-openmp 0.3.20+ds-1`
  - abbreviation: `openblas` or `oblas`

- Intel oneAPI Math Kernel Library (oneMKL):
BLAS/LAPACK combo libraries

- **OpenBLAS:**
  - based on GotoBLAS2 written by Kazushige Gotô
  - version: 0.3.20
  - package used: `libopenblas0-openmp 0.3.20+ds-1`
  - abbreviation: `openblas` or `oblas`

- **Intel oneAPI Math Kernel Library (oneMKL):**
  - version: 2023.1 build 20230303
  - package used: `intel-oneapi-mkl-2023.1.0`
BLAS/LAPACK combo libraries

- **OpenBLAS:**
  - based on GotoBLAS2 written by Kazushige Gotō
  - version: 0.3.20
  - package used: `libopenblas0-openmp 0.3.20+ds-1`
  - abbreviation: `openblas` or `oblas`

- **Intel oneAPI Math Kernel Library (oneMKL):**
  - version: 2023.1 build 20230303
  - package used: `intel-oneapi-mkl-2023.1.0`
  - abbreviation: `mkl`
Kernels used

Values reported by `$sysinfo["blascore"]`

<table>
<thead>
<tr>
<th>library</th>
<th>AMD</th>
<th>Intel</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenBLAS</td>
<td>Zen</td>
<td>SkylakeX</td>
</tr>
<tr>
<td>BLIS</td>
<td>zen3</td>
<td>haswell</td>
</tr>
<tr>
<td>AOCL-BLIS</td>
<td>zen3</td>
<td>—</td>
</tr>
<tr>
<td>oneMKL</td>
<td>—</td>
<td>AVX-512</td>
</tr>
</tbody>
</table>
Open Source

- GCC
- Clang/LLVM
Open Source

- GCC
  - version: 12.1.0
  - package used: 12.1.0-2ubuntu1~22.04

- Clang/LLVM
Open Source

- **GCC**
  - version: 12.1.0
  - package used: 12.1.0-2ubuntu1~22.04
  - abbreviation: gcc

- **Clang/LLVM**
Open Source

- **GCC**
  - version: 12.1.0
  - package used: `12.1.0-2ubuntu1~22.04`
  - abbreviation: `gcc`

- **Clang/LLVM**
  - based upon LLVM 15.0.7
Open Source

- **GCC**
  - version: 12.1.0
  - package used: 12.1.0-2ubuntu1~22.04
  - abbreviation: gcc

- **Clang/LLVM**
  - based upon LLVM 15.0.7
  - version: Ubuntu Clang 15.0.7
  - package used: 1:15.0.7-0ubuntu0.22.04.1
Open Source

- **GCC**
  - version: 12.1.0
  - package used: `12.1.0-2ubuntu1~22.04`
  - abbreviation: `gcc`

- **Clang/LLVM**
  - based upon LLVM 15.0.7
  - version: Ubuntu Clang 15.0.7
  - package used: `1:15.0.7-0ubuntu0.22.04.1`
  - abbreviation: `llvm`
Provided by CPU vendor

- AMD Optimizing C/C++ and Fortran Compilers (AOCC)

- Intel oneAPI DPC++/C++ Compiler
- AMD Optimizing C/C++ and Fortran Compilers (AOCC)
  - based on LLVM Mirror.Version.14.0.6

- Intel oneAPI DPC++/C++ Compiler
- AMD Optimizing C/C++ and Fortran Compilers (AOCC)
  - based on LLVM Mirror. Version 14.0.6
  - version: AMD Clang 14.0.6 (CLANG: AOCC_4.0.0-Build#434 2022_10_28)
  - package used: aocc-compiler-4.0.0

- Intel oneAPI DPC++/C++ Compiler
Provided by CPU vendor

- AMD Optimizing C/C++ and Fortran Compilers (AOCC)
  - based on LLVM Mirror.Version.14.0.6
  - version: AMD Clang 14.0.6 (CLANG: AOCC_4.0.0-Build#434 2022_10_28)
  - package used: aocc-compiler-4.0.0
  - abbreviation: aocc

- Intel oneAPI DPC++/C++ Compiler
Provided by CPU vendor

- AMD Optimizing C/C++ and Fortran Compilers (AOCC)
  - based on LLVM Mirror. Version. 14.0.6
  - version: AMD Clang 14.0.6 (CLANG: AOCC_4.0.0-Build#434 2022_10_28)
  - package used: aocc-compiler-4.0.0
  - abbreviation: aocc

- Intel oneAPI DPC++/C++ Compiler
  - based upon LLVM 16.0.0git
Provided by CPU vendor

- **AMD Optimizing C/C++ and Fortran Compilers (AOCC)**
  - based on LLVM Mirror.Version.14.0.6
  - version: AMD Clang 14.0.6 (CLANG: AOCC_4.0.0-Build#434 2022_10_28)
  - package used: aocc-compiler-4.0.0
  - abbreviation: aocc

- **Intel oneAPI DPC++/C++ Compiler**
  - based upon LLVM 16.0.0git
  - version: Intel oneAPI DPC++/C++ Compiler 2023.1.0 (2023.1.0.20230320)
  - package used: intel-oneapi-dpcpp-cpp-2023.1.0
Provided by CPU vendor

- **AMD Optimizing C/C++ and Fortran Compilers (AOCC)**
  - based on LLVM Mirror. Version 14.0.6
  - version: AMD Clang 14.0.6 (CLANG: AOCC_4.0.0-Build#434 2022_10_28)
  - package used: aocc-compiler-4.0.0
  - abbreviation: aocc

- **Intel oneAPI DPC++/C++ Compiler**
  - based upon LLVM 16.0.0git
  - version: Intel oneAPI DPC++/C++ Compiler 2023.1.0 (2023.1.0.20230320)
  - package used: intel-oneapi-dpcpp-cpp-2023.1.0
  - abbreviation: icx
We use *gretl* version 2023a compiled with the following flags:

```
-march=native -mtune=native -O3.
```
We use *gretl* version 2023a compiled with the following flags:

\[-\text{march=native} \ \text{-mtune=native} \ -O3.\]

In case of compilers provided by CPU vendor we also use:

<table>
<thead>
<tr>
<th></th>
<th>aocl</th>
<th>mkl</th>
<th>oblas</th>
<th>bflame</th>
</tr>
</thead>
<tbody>
<tr>
<td>aocc</td>
<td>-ffast-math</td>
<td>—</td>
<td></td>
<td></td>
</tr>
<tr>
<td>icx</td>
<td>—</td>
<td>—</td>
<td>-fp-model=precise</td>
<td></td>
</tr>
</tbody>
</table>
We always use: `-l:libflame.so.4.0 -lblis-mt`. 
We always use: `\texttt{-l:libflame.so.4.0 -lblis-mt.}`.

In case of different compilers we also use:

- `\texttt{gcc -lm -ldl}`
- `\texttt{llvm -fopenmp=libomp}`
- `\texttt{aocc -fopenmp=libomp -lalm}`
We always use:

-L/opt/intel/oneapi/mkl/2023.1.0/lib/intel64
-lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core
-liomp5 -lpthread -lm -ldl.
We always use: \texttt{-lopenblas}.\"
We always use: -lflame -lblis -ldl.
In simulations we use:

1. **matrix_perf** (ver. 1.11) by Allin Cottrell
2. **DCC** (ver. 0.1) by Riccardo "Jack" Lucchetti, Giulio Palomba and Luca Pedini
3. **StrucTiSM** (ver. 0.7) by Riccardo "Jack" Lucchetti and Sven Schreiber
4. **BMA** (ver. 3.1) by Marcin Błażejowski and Jacek Kwiatkowski
General assumptions

Each simulation was repeated 10 times. Before each simulation (but right after setting) we put it to sleep for 3 seconds. Each scenario is run in non-threaded and threaded mode. For the later we use:

```
set omp_num_threads $sysinfo["ncores"]
```

We have scenarios for smaller and bigger cases which is controlled via different $mnk$ values.

Which C compiler and BLAS/LAPACK library...
General assumptions

- Each simulation was repeated 10 times.
General assumptions

- Each simulation was repeated 10 times.
- Before each simulation (but right after setting) we put it to **sleep** for 3 seconds.
Each simulation was repeated 10 times.

Before each simulation (but right after setting) we put it to **sleep** for 3 seconds.

Each scenario is run in non-threaded and threaded mode. For the later we use:

```c
set omp_num_threads $sysinfo["ncores"]
```
General assumptions

- Each simulation was repeated 10 times.
- Before each simulation (but right after setting) we put it to `sleep` for 3 seconds.
- Each scenario is run in non-threaded and threaded mode. For the later we use:
  ```bash
  set omp_num_threads $sysinfo["ncores"].
  ```
- We have scenarios for smaller and bigger cases which is controlled via different `mnk` values.
We use matrix_perf with Allin's setup. The package was slightly modified to return results as bundle of bundles of matrices.
We use matrix_perf with Allin’s setup.
We use `matrix_perf` with Allin’s setup.

The packages was slightly modified to return results as `bundle` of bundles of matrixes.
LAPACK/BLAS test

<table>
<thead>
<tr>
<th>Environment</th>
<th>Compilation</th>
<th>Simulation scenarios</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>dgemm efficiency (pure BLAS Level 3)</td>
<td>LAPACK/BLAS efficiency</td>
<td>Real econometrics efficiency</td>
<td></td>
</tr>
</tbody>
</table>

We use the following build-in functions: mols, qrdecomp, cholesky, inv, invpd, eigen, det, psdroot, svd.

Dimensions for smaller cases:
- non-square: 5000 × 40 (m = k = 5000, n = 40)
- square: 500 × 500 (m = n = k = 500)

Dimensions for bigger cases:
- non-square: 5000 × 200 (m = k = 5000, n = 200)
- square: 1500 × 1500 (m = n = k = 1500)

Marcin Błażejowski

Which C compiler and BLAS/LAPACK library...
LAPACK/BLAS test

- We use the following build-in functions: mols, qrdecomp, cholesky, inv, invpd, eigen, det, psdroot, svd
LAPACK/BLAS test

- We use the following build-in functions: mols, qrdecomp, cholesky, inv, invpd, eigen, det, psdroot, svd

- Dimensions for smaller cases:

- Dimensions for bigger cases:
LAPACK/BLAS test

- We use the following build-in functions: \texttt{mols, qrdecomp, cholesky, inv, invpd, eigen, det, psdroot, svd}
- Dimensions for smaller cases:
  - non-square: 5000 \times 40 (m = k = 5000, n = 40)
  - square: 500 \times 500 (m = n = k = 500)
- Dimensions for bigger cases:
LAPACK/BLAS test

- We use the following build-in functions: mols, qrdecomp, cholesky, inv, invpd, eigen, det, psdroot, svd
- Dimensions for smaller cases:
  - non-square: 5000 × 40 (m = k = 5000, n = 40)
  - square: 500 × 500 (m = n = k = 500)
- Dimensions for bigger cases:
  - non-square: 5000 × 200 (m = k = 5000, n = 200)
  - square: 1500 × 1500 (m = n = k = 1500)
<table>
<thead>
<tr>
<th>Environment</th>
<th>Compilation</th>
<th>Simulation scenarios</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>dgemm efficiency (pure BLAS Level 3)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LAPACK/BLAS efficiency</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Real econometrics efficiency</td>
<td></td>
</tr>
</tbody>
</table>

**gretl’s packages test**

Dimensions for smaller cases:
- **DCC**: $528 \times 3$ ($m = k = 528$, $n = 3$)
- **StrucTiSM**: $731 \times 4$ ($m = k = 731$, $n = 4$)
- **BMA**: $72 \times 14$ ($m = k = 72$, $n = 14$)

Dimensions for bigger cases:
- **DCC**: $528 \times 4$ ($m = k = 528$, $n = 4$)
- **StrucTiSM**: $731 \times 8$ ($m = k = 731$, $n = 8$)
- **BMA**: $72 \times 41$ ($m = k = 72$, $n = 41$)

Marcin Błażejowski

Which C compiler and BLAS/LAPACK library...
gretl’s packages test

- Dimensions for smaller cases:

- Dimensions for bigger cases:
gretl’s packages test

- Dimensions for smaller cases:
  - DCC: $528 \times 3 \ (m = k = 528, n = 3)$
  - StrucTiSM: $731 \times 4 \ (m = k = 731, n = 4)$
  - BMA: $72 \times 14 \ (m = k = 72, n = 14)$

- Dimensions for bigger cases:

Marcin Błażejowski

Which C compiler and BLAS/LAPACK library...
gretl’s packages test

- Dimensions for smaller cases:
  - DCC: $528 \times 3 \ (m = k = 528, n = 3)$
  - StrucTiSM: $731 \times 4 \ (m = k = 731, n = 4)$
  - BMA: $72 \times 14 \ (m = k = 72, n = 14)$

- Dimensions for bigger cases:
  - DCC: $528 \times 4 \ (m = k = 528, n = 4)$
  - StrucTiSM: $731 \times 8 \ (m = k = 731, n = 8)$
  - BMA: $72 \times 41 \ (m = k = 72, n = 41)$
Figure: Results of `matrix_perf_1_1` experiment: overall setup: $m = 128, n = 128, k = 128 \ldots 2014$
Figure: Results of `matrix_perf_1_1` experiment: library specific

setup: \( m = 128, n = 128, k = 128 \ldots 2014 \)
Environment
Compilation
Simulation scenarios

Results

dgemm efficiency (pure BLAS Level 3)
LAPACK/BLAS efficiency
Real econometrics efficiency
Overall performance

Figure: Results of matrix_perf_1_1 experiment: compiler specific
setup: \( m = 128, n = 128, k = 128 \ldots 2014 \)
Results

dgemm efficiency (pure BLAS Level 3)
LAPACK/BLAS efficiency
Real econometrics efficiency
Overall performance

(a) AMD
(b) Intel

Figure: Results of matrix_perf_1_2 experiment: overall setup: $m = 128 \ldots 2048$, $n = 128 \ldots 2048$, $k = 128$
Figure: Results of matrix_perf_1_2 experiment: library specific setup: $m = 128 \ldots 2048$, $n = 128 \ldots 2048$, $k = 128$
Environment
Compilation
Simulation scenarios
Results
dgemm efficiency (pure BLAS Level 3)
LAPACK/BLAS efficiency
Real econometrics efficiency
Overall performance

Marcin Błażejowski
Which C compiler and BLAS/LAPACK library...
Figure: Results of matrix_perf_1_3 experiment: overall setup: $m = 128 \ldots 2048$, $n = 128 \ldots 2048$, $k = 128 \ldots 2048$
Marcin Błażejowski

Which C compiler and BLAS/LAPACK library...
Figure: Results of `matrix_perf_1_3` experiment: compiler specific setup: $m = 128 \ldots 2048$, $n = 128 \ldots 2048$, $k = 128 \ldots 2048$
Figure: Results of `matrix_perf_2_1` experiment: overall setup: $m = 8 \ldots 4096, n = 8, k = 8$
Figure: Results of matrix_perf_2_1 experiment: library specific setup: \( m = 8 \ldots 4096, n = 8, k = 8 \)
Figure: Results of matrix_perf_2_1 experiment: compiler specific

setup: $m = 8 \ldots 4096, n = 8, k = 8$
Figure: Results of `matrix_perf_2_2` experiment: overall setup: \( m = 10 \ldots 5120, n = 2, k = 1000 \)
Figure: Results of matrix_perf_2_2 experiment: library specific setup: \( m = 10 \ldots 5120, n = 2, k = 1000 \)
Figure: Results of matrix_perf_2_2 experiment: compiler specific setup: \( m = 10 \ldots 5120, n = 2, k = 1000 \)
Figure: Results of `matrix_perf_2_3` experiment: overall setup: $m = 10 \ldots 320, n = 10, k = 1000$
Environment
Compilation
Simulation scenarios
Results
dgemm efficiency (pure BLAS Level 3)
LAPACK/BLAS efficiency
Real econometrics efficiency
Overall performance

(a) AMD
(b) Intel

Figure: Results of matrix_perf_2_3 experiment: library specific
setup: $m = 10 \ldots 320, n = 10, k = 1000$
Environment
Compilation
Simulation scenarios
Results
dgemm efficiency (pure BLAS Level 3)
LAPACK/BLAS efficiency
Real econometrics efficiency
Overall performance

Figure: Results of matrix_perf_2_3 experiment: compiler specific
setup: $m = 10 \ldots 320, n = 10, k = 1000$
Figure: Results of algebra_perf_0_1 experiment: overall setup: \( m = 5000, n = 40 \) or \( m = n = 500 \) (non-threaded)
Environment
Compilation
Simulation scenarios

Results

dgemm efficiency (pure BLAS Level 3)
LAPACK/BLAS efficiency
Real econometrics efficiency
Overall performance

Figure: Results of algebra_perf_0_1 experiment: library specific

setup: $m = 5000$, $n = 40$ or $m = n = 500$ (non-threaded)
Environment
Compilation
Simulation scenarios
Results
dgemm efficiency (pure BLAS Level 3)
LAPACK/BLAS efficiency
Real econometrics efficiency
Overall performance

Marcin Błażejowski
Which C compiler and BLAS/LAPACK library...
Figure: Results of `algebra_perf_0_2` experiment: overall setup: $m = 5000, n = 200$ or $m = n = 1500$ (non-threaded)
Figure: Results of `algebra_perf_0_2` experiment: library specific setup: $m = 5000, n = 200$ or $m = n = 1500$ (non-threaded)
Figure: Results of algebra_perf_0_2 experiment: compiler specific setup: \( m = 5000, n = 200 \) or \( m = n = 1500 \) (non-threaded)
Figure: Results of `algebra_perf_1_1` experiment: overall setup: $m = 5000, n = 40$ or $m = n = 500$ (threaded)
Environment
Compilation
Simulation scenarios
Results
dgemm efficiency (pure BLAS Level 3)
LAPACK/BLAS efficiency
Real econometrics efficiency
Overall performance

Figure: Results of algebra_perf_1_1 experiment: library specific
setup: \(m = 5000, n = 40\) or \(m = n = 500\) (threaded)
Figure: Results of algebra_perf_1_1 experiment: compiler specific setup: $m = 5000, n = 40$ or $m = n = 500$ (threaded)
Environment
Compilation
Simulation scenarios
Results
dgemm efficiency (pure BLAS Level 3)
LAPACK/BLAS efficiency
Real econometrics efficiency
Overall performance

Figure: Results of algebra_perf_1_2 experiment: overall setup: $m = 5000, n = 200$ or $m = n = 1500$ (threaded)
Figure: Results of algebra_perf_1_2 experiment: library specific
setup: $m = 5000, n = 200$ or $m = n = 1500$ (threaded)
Environment
Compilation
Simulation scenarios
Results
dgemm efficiency (pure BLAS Level 3)
LAPACK/BLAS efficiency
Real econometrics efficiency
Overall performance

Figure: Results of algebra_perf_1_2 experiment: compiler specific setup: \( m = 5000, n = 200 \) or \( m = n = 1500 \) (threaded)
Figure: Results for selected packages experiment: overall setup: smaller cases (non-threaded)
**Results**

**(a) AMD**

**(b) Intel**

**Figure:** Results for selected packages experiment: library specific setup: smaller cases (non-threaded)
Figure: Results for selected packages experiment: compiler specific setup: smaller cases (non-threaded)
Results

dgemm efficiency (pure BLAS Level 3)
LAPACK/BLAS efficiency
Real econometrics efficiency
Overall performance

Environment
Compilation
Simulation scenarios

(b) Intel

Figure: Results for selected packages experiment: overall setup: bigger cases (non-threaded)
Environment
Compilation
Simulation scenarios
Results
dgemm efficiency (pure BLAS Level 3)
LAPACK/BLAS efficiency
Real econometrics efficiency
Overall performance

Figure: Results for selected packages experiment: library specific
setup: bigger cases (non-threaded)
Figure: Results for selected packages experiment: compiler specific setup: bigger cases (non-threaded)
Figure: Results for selected packages experiment: overall setup: smaller cases (threaded)
Figure: Results for selected packages experiment: library specific setup: smaller cases (threaded)
Figure: Results for selected packages experiment: compiler specific setup: smaller cases (threaded)
Figure: Results for selected packages experiment: overall setup: bigger cases (threaded)
**Results**

![Graphs showing efficiency for AMD and Intel](image)

**Figure:** Results for selected packages experiment: library specific setup: bigger cases (threaded)

Marcin Błażejowski
Figure: Results for selected packages experiment: compiler specific setup: bigger cases (threaded)
To calculate overall performance (across all three scenarios) we use *Min-Max Feature Scaling*:
To calculate overall performance (across all three scenarios) we use *Min-Max Feature Scaling*:

- for results in Gflops:

\[
x' = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}
\]
To calculate overall performance (across all three scenarios) we use *Min-Max Feature Scaling*:

- for results in Gflops:

\[ x' = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} \]

- for results in seconds:

\[ x' = \frac{x_{\text{max}} - x}{x_{\text{max}} - x_{\text{min}}} \]
Figure: Overall score based on three experiments: compiler specific

(a) AMD

(b) Intel
Figure: Overall score based on three experiments: library specific
The end