Close Coupling of AOCL-BLAS in AOCL-LAPACK

Sridhar G
Pradeep Rao
Vasantha Kumar R

AMD
together we advance_

[Public]

AGENDA

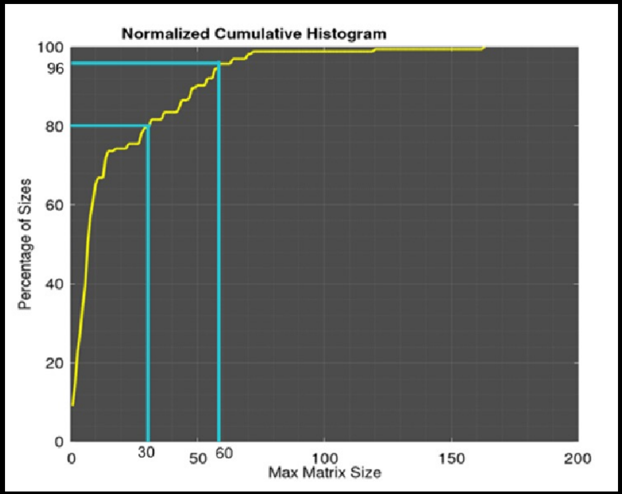
- Introduction
- Motivation
- Observation
- Proposed Solution
- Results
- Challenges
- Q & A

Introduction

- AOCL (AMD optimizing CPU libraries) is AMD's CPU Math Library, tuned for AMD processor
- AOCL-LAPACK, a fork of libFLAME library (repository from UT Austin), is optimized as part of AOCL
- AOCL-LAPACK depends on AOCL-BLAS (a fork of BLIS library optimized as part of AOCL) for various low-level vector and matrix operations
- Our last public release is AOCL 4.2:
  - Improved performance of GESVD, GETRF, GETRS and GESV
  - OpenMP parallelism enabled in {c,z}hetrd, {s,d}sytrd_sb2st and iparam2stage
  - Option to link with AOCL-BLAS during build to enable invoking AOCL_BLAS internal APIs
  - Cmake improvements
  - Test suite framework enhancement
- Latest AOCL-LAPACK source code is available on github: https://github.com/amd/libflame

Motivation (1/2)

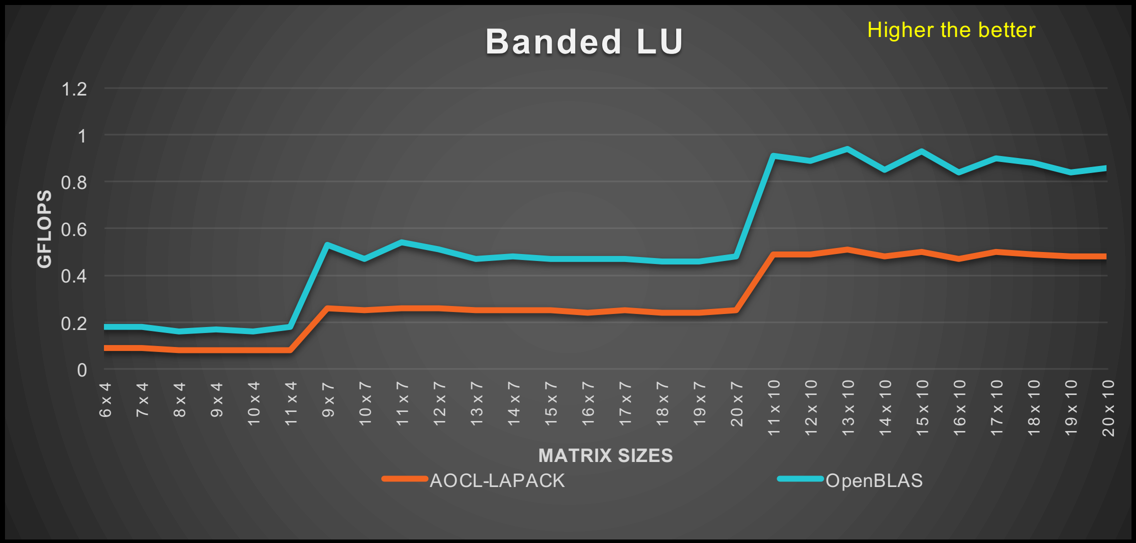
- Performance of AOCL-LAPACK is competitive with OpenBLAS for medium and large matrix inputs, especially with well tuned AOCL-BLAS library
- OpenBLAS is an optimized open-source implementation of BLAS and LAPACK libraries
- Problem sizes commonly encountered in many of our client benchmarks are small
- Graph depicts the problem sizes for DGBTRF API used in one of these benchmarks



AMD
together we advance_

Motivation (2/2)

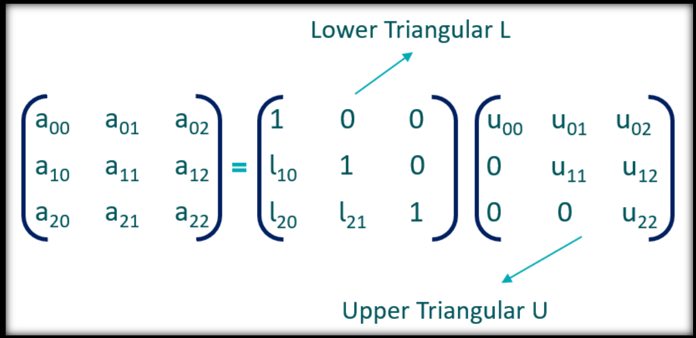
How is the performance of DGBTRF for these sizes?



- We are, on an, average 50% behind openBLAS for small matrix sizes

AMD
together we advance_

LU Decomposition



❖ Definition of LU for matrix A:

$A = L * U$ where, L – Lower triangular matrix
U – Upper triangular matrix

❖ Operations in LU for each column:

(1) Partition input A as in top part of figure

(2) Update $l_{21} = a_{21}/\alpha_{11}$ (*DSCAL*)

(3) $u_{11} = \alpha_{11}$, $u_{12}^T = a_{12}^T$

(4) Update $A'_{22} = A_{22} - l_{21} * u_{12}^T$ (*DGER*)

(5) $A = A'_{22}$ and repeat steps (1) to (5)

$$A = \left(\frac{\alpha_{11}}{a_{21}} \middle| \frac{a^T{}_{12}}{A_{22}}\right)$$

$$L = \left(\frac{1}{l_{21}} \middle| \frac{0}{L_{22}}\right) \qquad U = \left(\frac{u_{11}}{0} \middle| \frac{u^T{}_{12}}{U_{22}}\right)$$

AMD
together we advance_

Banded LU Decomposition (GBTRF)

- GBTRF is a variant of LU factorization API in LAPACK, which performs factorization on banded storage matrix

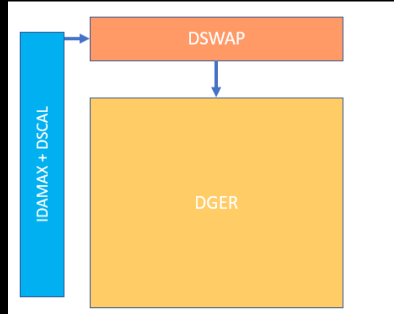
- **Advantages:**
  - **Space Efficiency:** Only the non-zero elements are stored, which significantly reduces memory usage for large matrices

  - **Performance:** Accessing elements within the band is faster due to the compact storage

- Column wise computation involves:
  - Pivoting and scaling: IDAMAX() + DSCAL()
  - Row swapping: DSWAP()
  - Rank-1 operation: DGER()

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & a_{34} & 0 \\ 0 & a_{22} & a_{23} & a_{22} & a_{23} \\ 0 & 0 & a_{33} & a_{33} & a_{34} \end{bmatrix} \longrightarrow \begin{bmatrix} * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{bmatrix}$$

Band matrix → *Band storage matrix*

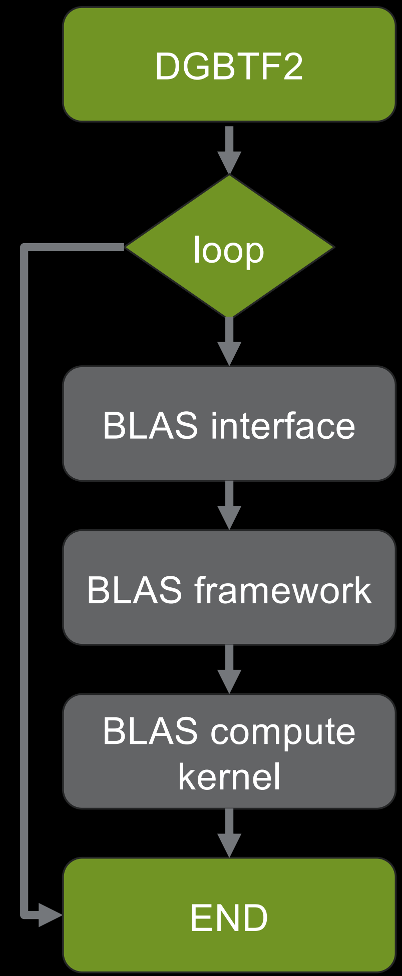


LU Factorization
(Blue arrows represent the flow of algorithm)

AMD
together we advance_

Observation (1/2)

- Existing design scales well for medium and large matrix inputs but shows significant performance degradation for small matrix inputs
- Although the relevant AOCL-BLAS APIs are optimized for small sizes, LAPACK layer was unable to fully leverage this optimization
- Profiling revealed around 30% to 40% of overhead was from AOCL-BLAS framework related functions
- When AOCL-BLAS API is called, a series of framework related functions are executed before the actual computational kernel runs
- These framework functions are essential for selecting the appropriate kernel based on factors such as machine instruction support and problem size



AMD
together we advance_

Observation (2/2)

- LU factorization is an iterative algorithm, and the computation of each column involves calling the same set of AOCL-BLAS APIs
- For small matrix inputs, the repeated overhead from framework functions associated with each AOCL-BLAS API call significantly affect the overall performance of the API
- Our goal was to minimize the overhead from the framework functions as much as possible to improve the performance

[Public]

Proposed solution 1

Inline the BLAS compute kernels directly into AOCL-LAPACK

- Solution Steps
  - Implement / Inline all functions in LAPACK itself → Completely eliminates the function call overheads
  - Remove the framework overheads → Since the range of sizes are limited, a single optimized kernel can be chosen

Drawbacks:

- It is a challenging task to apply across the library
- This solution may lead to code fragmentation – must be used only if required

AMD
together we advance_

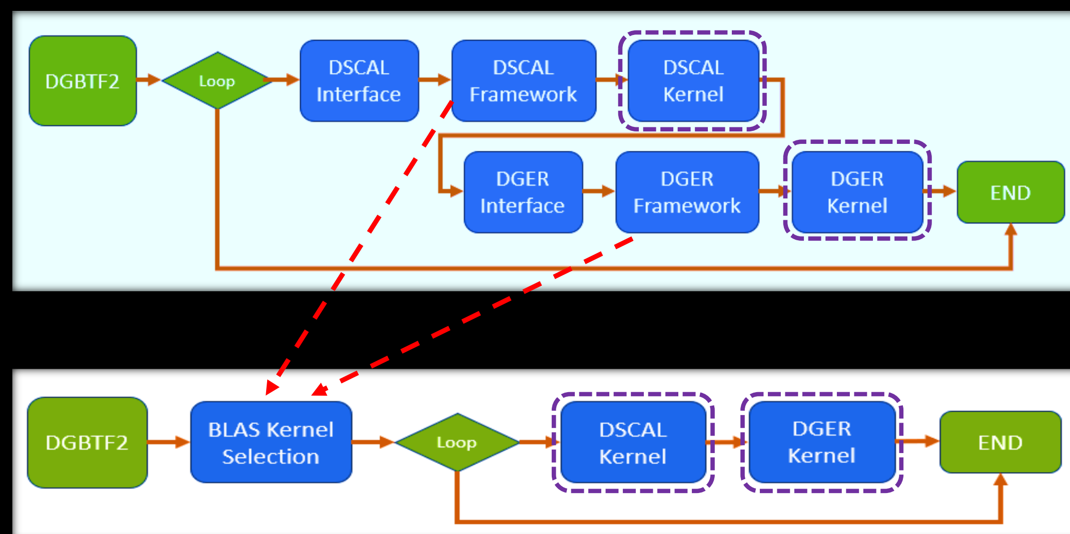
[Public]

Proposed Solution 2 (1/2)

- Instead of calling the standard interface of AOCL-BLAS APIs, we propose invoking the corresponding compute kernels directly from AOCL-LAPACK
- Framework to choose the right AOCL-BLAS compute kernel will now be executed within AOCL-LAPACK layer
- AOCL-BLAS Framework code to select the appropriate compute kernels will now be executed only once per AOCL-LAPACK API call

Original Code Flow:

Optimized Code Flow:



[Public]

Proposed Solution 2 (2/2)

- We observed significant reduction in the number of instruction executed per AOCL-BLAS API call, with an average reduction of 60% to 80%

BLAS API	Instruction executed per call		Instruction difference
	Standard interface	Kernel Interface	
DGER	2056	306	-85.11%
IDAMAX	4734	579	-87.76%
DSCAL	480	165	-65.62%
DSWAP	385	71	-81.55%

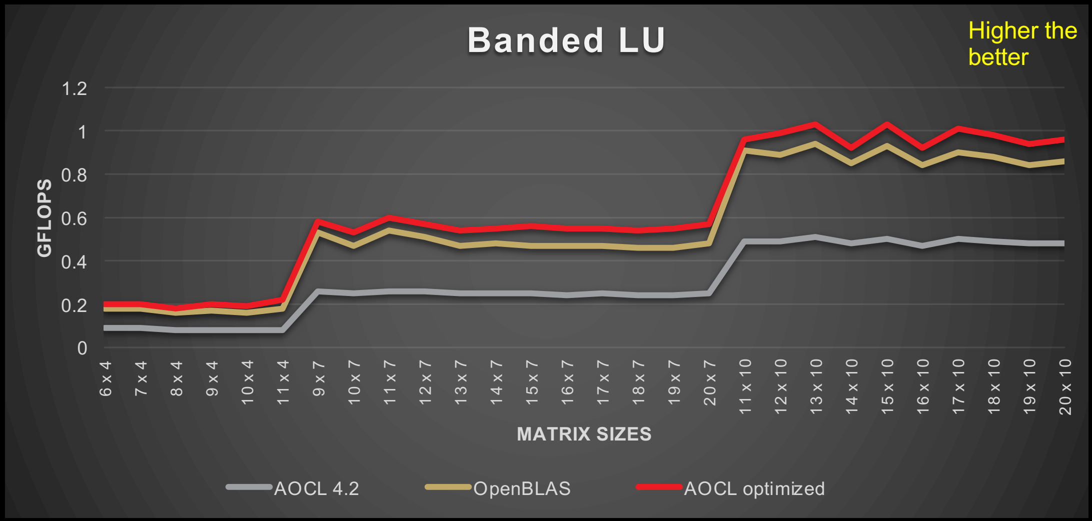
- This solution allows LAPACK to fully leverage the computational power of AOCL-BLAS APIs, particularly for small input sizes

- From an Implementation perspective, it is relatively straightforward, involving only the retrieval and invocation of appropriate AOCL-BLAS compute kernels

- We were able to achieve better performance while preserving the modularity between the libraries

AMD
together we advance_

Results



- Performance improved on an average by 110%
- AOCL-LAPACK now outperforms OpenBLAS for small matrix sizes

AMD
together we advance_

Challenges

- Not all AOCL-BLAS compute kernels, and framework functions are exposed. We must ensure all the relevant code is open to AOCL-LAPACK
- Any kernel addition or modification must be appropriately updated within the AOCL-BLAS framework context, as AOCL-LAPACK is reliant on it for kernel selection

Questions

[Public]

COPYRIGHT AND DISCLAIMER

AMD
together we advance_

AMD