Validation and evaluation of the Chameleon Lapack interface

Second year internship

Alycia :)
What is Chameleon?

- Dense Linear Algebra Library
  - LAPACK (OpenBlas, Intel MKL) Multi-Thread
  - SCALAPACK (Netlib, Intel MKL) MPI
- Parallel → MPI, PThread, CUDA
- Task based
Chameleon Algorithms

**MM**
\[ C = \alpha A x B + \beta C \]

**RK**
\[ C = \alpha A x A^T + \beta C \]

**R2K**
\[ C = \alpha A x B^T + \alpha A^T x B + \beta C \]

BLAS3
Chameleon Algorithms

**MM**
\[ C = \alpha A \times B + \beta C \]

**RK**
\[ C = \alpha A \times A^T + \beta C \]

**R2K**
\[ C = \alpha A \times B^T + \alpha A^T \times B + \beta C \]

**TRF**
\[ A = L \times U \]
\[ A = L \times L' \]
\[ A = U^T \times U \]

**Cholesky** / **LU Decomposition**

**SV** / **TRS**
\[ A \times X = B \]

**Tri**
\[ A^{-1} \]
Chameleon Algorithms

**QR / LQ Factorisation**

<table>
<thead>
<tr>
<th>Method</th>
<th>Factorisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM</td>
<td>$C = \alpha A x B + \beta C$</td>
</tr>
<tr>
<td>RK</td>
<td>$C = \alpha A x A^T + \beta C$</td>
</tr>
<tr>
<td>R2K</td>
<td>$C = \alpha A x B^T + \alpha A^T x B + \beta C$</td>
</tr>
</tbody>
</table>

**BLAS3**

**QR**

<table>
<thead>
<tr>
<th>$A$</th>
<th>$= Q x R$</th>
</tr>
</thead>
</table>

**LQ**

<table>
<thead>
<tr>
<th>$A$</th>
<th>$= L x Q$</th>
</tr>
</thead>
</table>

**CHOLESKY / LU Decomposition**

<table>
<thead>
<tr>
<th>$A$</th>
<th>$= L x U$</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>$A$</th>
<th>$= L x L'$</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>$A$</th>
<th>$= U x U^T$</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>$A$</th>
<th>$= A^{-1}$</th>
</tr>
</thead>
</table>

**MM**

| $C = \alpha A x B + \beta C$ |

**RK**

| $C = \alpha A x A^T + \beta C$ |

**R2K**

| $C = \alpha A x B^T + \alpha A^T x B + \beta C$ |

**TRF**

| $A = L x U$ |

| $A = L x L'$ |

| $A = U x U^T$ |

| $A = A^{-1}$ |

**SV / TRS**

| $A x X = B$ |
Chameleon Algorithms

QR / LQ Factorisation
- MM: \( C = \alpha A \times B + \beta C \)
- RK: \( C = \alpha A \times A^T + \beta C \)
- R2K: \( C = \alpha A \times B^T + \alpha A^T \times B + \beta C \)
- TRF: \( A = L \times U \)

BLAS3

Singular Value Decomposition
- SVD: \( A = U \times \Sigma \times V^T \)

Cholesky / LU Decomposition
- TRF: \( A = L \times \) (with transpose)
- SV / TRS: \( A \times X = B \)
- TRI: \( A^{-1} = U \times U^T \)
Chameleon Algorithms

**MM**
\[ C = \alpha A x B + \beta C \]

**RK**
\[ C = \alpha A x A^T + \beta C \]

**R2K**
\[ C = \alpha A x B^T + \alpha A^T x B + \beta C \]

**TRF**
\[ A = L x U \]
\[ A = L x L^T \]
\[ A = \text{TRI } A^{-1} \]

**QR / LQ Factorisation**
\[ A = Q x R \]
\[ A = L x Q \]

**BLAS3**

**CHOLESKY / LU Decomposition**
\[ A = L x U \]
\[ A x X = B \]

**Singular Value Decomposition**
\[ A = U x L x Q \]

**SVD**

**QR**

**LQ**

**LU Decomposition**

**SVD**
Task-based Algorithm: POTRF

Chameleon_potrf( A )
    for j = 0 to N-1 do
        potrf( A_{RW}[j][j] )
        for i = j+1 to N-1 do
            trsm( A_{RW}[i][j], A_R[j][j] )
        done
        for i = j+1 to N-1 do
            syrk( A_{RW}[i][i], A_R[i][j] )
            for k = j+1 to i-1 do
                gemm( A_{RW}[i][k], A_R[i][j], A_R[k][j] )
            done
        done
    done
Task-based Algorithm: POTRF

Chameleon_potrf( A )

for j = 0 to N-1 do
    potrf( A_{RW}[j][j] )
    for i = j+1 to N-1 do
        trsm( A_{RW}[i][j], A_{R}[j][j] )
        done
    for i = j+1 to N-1 do
        syrk( A_{RW}[i][i], A_{R}[i][j] )
        for k = j+1 to i-1 do
            gemm( A_{RW}[i][k], A_{R}[i][j], A_{R}[k][j] )
            done
        done
    done
done
Task-based Algorithm: POTRF

Chameleon_potrf( A )
for j = 0 to N-1 do
  \texttt{potrf}( A_{RW}[j][j] )
  for i = j+1 to N-1 do
    \texttt{trsm}( A_{RW}[i][j], A_R[j][j] )
    done
  for i = j+1 to N-1 do
    \texttt{syrk}( A_{RW}[i][i], A_R[i][j] )
    for k = j+1 to i-1 do
      \texttt{gemm}( A_{RW}[i][k], A_R[i][j], A_R[k][j] )
    done
  done
done
Task-based Algorithm: POTRF

Chameleon_potrf( A )
  for j = 0 to N-1 do
    potrf( A_{RW}[j][j] )
    for i = j+1 to N-1 do
      trsm( A_{RW}[i][j], A_R[j][j] )
    done
  for i = j+1 to N-1 do
    syrk( A_{RW}[i][i], A_R[i][j] )
    for k = j+1 to i-1 do
      gemm( A_{RW}[i][k], A_R[i][j], A_R[k][j] )
    done
  done
  done

10
Task-based Algorithm: POTRF

Chameleon_potrf( A )
for j = 0 to N-1 do
  potrf( A_{RW}[j][j] )
  for i = j+1 to N-1 do
    trsm( A_{RW}[i][j], A_R[j][j] )
  done
for i = j+1 to N-1 do
  syrk( A_{RW}[i][i], A_R[i][j] )
  for k = j+1 to i-1 do
    gemm( A_{RW}[i][k], A_R[i][j], A_R[k][j] )
  done
done
done
Task-based Algorithm: **POTRF**

```plaintext
Chameleon_potrf( A )
  for j = 0 to N-1 do
    potrf( A_RW[j][j] )
    for i = j+1 to N-1 do
      trsm( A_RW[i][j], A_R[j][j] )
    done
  for i = j+1 to N-1 do
    syrk( A_RW[i][i], A_R[i][j] )
    for k = j+1 to i-1 do
      gemm( A_RW[i][k], A_R[i][j], A_R[k][j] )
    done
  done
```

Task-based Algorithm: POTRF

Chameleon_potrf( A )
for j = 0 to N-1 do
  potrf( A_{RW}[j][j] )
  for i = j+1 to N-1 do
    trsm( A_{RW}[i][j], A_{R}[j][j] )
  done
for i = j+1 to N-1 do
  syrk( A_{RW}[i][i], A_{R}[i][j] )
  for k = j+1 to i-1 do
    gemm( A_{RW}[i][k], A_{R}[i][j], A_{R}[k][j] )
  done
done
done
done
Task-based Algorithm: POTRF

Chameleon_potrf( A )
for j = 0 to N-1 do
  potrf( A_RW[j][j] )
  for i = j+1 to N-1 do
    trsm( A_RW[i][j], A_R[j][j] )
done
for i = j+1 to N-1 do
  syrk( A_RW[i][i], A_R[i][j] )
  for k = j+1 to i-1 do
    gemm( A_RW[i][k], A_R[i][j], A_R[k][j] )
done
done
done
Task-based Algorithm: POTRF

Chameleon_potrf( A )
for j = 0 to N-1 do
    potrf( A_{RW}[j][j] )
    for i = j+1 to N-1 do
        trsm( A_{RW}[i][j], A_R[j][j] )
done
for i = j+1 to N-1 do
    syrk( A_{RW}[i][i], A_R[i][j] )
    for k = j+1 to i-1 do
        gemm( A_{RW}[i][k], A_R[i][j], A_R[k][j] )
done
done
done
Task-based Algorithm: POTRF

Chameleon_potrf( A )
  for j = 0 to N-1 do
    potrf( \( A_{RW}[j][j] \) )
    for i = j+1 to N-1 do
      trsm( \( A_{RW}[i][j], A_{R}[j][j] \) )
    done
  for i = j+1 to N-1 do
    syrk( \( A_{RW}[i][i], A_{R}[i][j] \) )
  for k = j+1 to i-1 do
    gemm( \( A_{RW}[i][k], A_{R}[i][j], A_{R}[k][j] \) )
  done
  done
  done
Chameleon Matrix Descriptor

LAPACK CM (Column Major)

CHAMELEON CCRB (Column Column Rectangular Block)

ALLOC-BY-TILE

ALLOC-GLOBAL
So what did I do?
Chameleon Interfaces

- Chameleon_zalgo
- Chameleon_zalgo_Tile
- Chameleon_zalgo_Tile_Async
- Synchronisation

- Chameleon_zalgo_Tile_Async
- Synchro
Chameleon Interfaces

- **Chameleon_zalgo**
  - No tests or checks made

- **Chameleon_zalgo_Tile**
  - Tested and checked since 2020 (commit n. 166) by L. Barros de Assis during his internship

- **Chameleon_zalgo_Tile_Async**
  - Tested indirectly since 2020 by Lucas
  - Tested directly since 2022 (commit n. 262)
Conversion **In-Place vs Out-Of-Place**

**In-Place**

$A_{CM}$

$DescA_{CM}$

**Out-Of-Place**

$DescA_{CCR}$

$Out-Of-Place$
### Performances: machines used

<table>
<thead>
<tr>
<th></th>
<th><strong>Number of Cores</strong></th>
<th><strong>Type of Core</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BORA</strong></td>
<td>2x 18</td>
<td>Intel CascadeLake</td>
</tr>
<tr>
<td><strong>ZONDA</strong></td>
<td>2x 32</td>
<td>AMD Zen2</td>
</tr>
<tr>
<td><strong>DIABLO</strong></td>
<td>2x 64</td>
<td>AMD Zen3</td>
</tr>
</tbody>
</table>
Performances

Chameleon

LAPACK

Library
- MKL Lapack interface
- Chameleon Lapack interface
Performances

**LAPACK-Layout**

**Conversion**
- In-Place
- Out-Of-Place

**Conversion Type**
- in-place
- out-of-place
Performances

Same behaviour with diablo

Opposite behaviour with zonda

Same behaviour with bora
Singular Value Decomposition

\[ A = U_1 \times \Sigma \times V^T \]

\[ A = U_1 \times U_2 \times \Sigma \times V^T \]

\[ A = U_1 \times U_2 \times U_3 \times \Sigma \times V^T \]

```
Chameleon_zgebrd_ge2gb
Lapacke_zgbbbrd
Lapacke_zdsqr
```

```
A
U
\Sigma
V^T
```
Singular Value Decomposition

Chameleon_zgebrd_ge2gb → Had minor errors
Needs to have the tree version

Chameleon_pztile2band → Was incorrect

Lapacke_zgbbbrd
Conclusion

Validation of the Standard API:
- Numerical ✓
- On CPU ✓
- On GPU ✓

Benchmark of the Lapack interface ✓

Singular Value Decomposition:
- Testing ✓
- Numerical validation of the singular values ✓
- Numerical validation of the singular vectors ⬤⬤⬤
- Improvement of the algorithm using trees ⬤⬤⬤
Françooiiis, j’ai une questiooonnnn