Workshop on Batched, Reproducible, and Reduced Precision BLAS
May 18th - 19th, 2016

Reference implementation and testing of batched BLAS routines

The University of Manchester
Innovative Computing Laboratory
University of Tennessee
and NLAFET
Motivations

Reference implementation
- Demonstrate that the specification is implementable
- Help to improve the specification
- Provide a reliable code to validate other implementations

Testing
- Valid calling sequences
- Correctness of the computed results

Remarks
- The reference implementation does not aim to be efficient
- This presentation focuses on Level 3 batched BLAS routines
1. Reference implementation

2. Accuracy testing

3. Experimental results

4. Concluding remarks
Outline

1. Reference implementation
2. Accuracy testing
3. Experimental results
4. Concluding remarks
Overview of the reference implementation

1. Calling sequences with respect to the specification

```c
dgemm_batch(const enum * transA, const enum * transB, const int * m,
    const int * n, const int * k, const double * alpha,
    const double * const double * arrayA, const int * lda,
    const double * const double * arrayB, const int * ldb,
    const double * beta, double **arrayC, const int * ldc,
    const int batch_count, const enum batch_opts, int * info)
```

- **batch_count**: style for the batched (BATCH_FIXED, BATCH_VARIABLE)
- **batch_opts**: number of sub-problems to be processed
- **info**: error handling array
Overview of the reference implementation

1. Calling sequences with respect to the specification
2. Arguments checking and error handling

Critical arguments

- **Fixed size case**: `exit` if any argument is incorrect
- **Variable size case**:
  - `Exit` for whole batch error (e.g. `batch_count`)
  - `Continue` for sub-problem error (e.g. matrix size)
- **Xerbla** for error handling
- **Flexible error handling strategy**
Overview of the reference implementation

1. Calling sequences with respect to the specification
2. Arguments checking and error handling
3. Call **reference** BLAS implementation to solve each sub-problem

```c
for (iter = 0; iter < batch_count; iter++)
{
    Call a reference BLAS routine
}
```

According to the specification, the reference implementation should provide results as accurate as reference BLAS routines
Outline

1. Reference implementation

2. Accuracy testing

3. Experimental results

4. Concluding remarks
Basic idea

- The quantity of interest is the relative error $err = \frac{\|C - \hat{C}\|}{\|C\|}$
- Typically we want $err \leq \tau \varepsilon$ where $\tau$ is some tolerance and $\varepsilon$ is the machine epsilon e.g. $\varepsilon = 10^{-16}$

Remarks

- The tolerance $\tau$ is selected independently for each problem, e.g. $\tau = 10, 30, 100$ in MAGMA/LAPACK etc
- The choice of $\tau = 10, 30 \ldots$ may not be very rigorous
- Investigate a rigorous approach for small BLAS problems?
Accuracy testing

Error analysis approach

- Perform error analysis of BLAS routines [Nicholas J. Higham, 2002]
- Use the **forward/backward error bound** for accuracy testing
  
  \[ \| \text{Forward}_\text{Error} \| \leq \| \text{Forward}_\text{Error}_\text{Bound} \| \text{ or } \| \text{Residual} \| \leq \| \text{Backward}_\text{Error}_\text{Bound} \| \]

Remarks

- No need of intensive experiments to set a tolerance \( \tau \)
- A rigorous error bound can seem pessimistic in general but meaningful for small BLAS problems
Accuracy testing

Forward error bound of GEMM

- Formula: \( C = \alpha AB + \beta C_{\text{init}} \)
- Forward error \( \| C - \hat{C} \|_{\infty} \)
- Forward error bound: \((N|\alpha|\|A\|_{\infty}\|B\|_{\infty} + |\beta|\|C_{\text{init}}\|_{\infty})\varepsilon\)

Remark

The computed solution \( \hat{C} \) is acceptable if

\[
\frac{\| C - \hat{C} \|_{\infty}}{(N|\alpha|\|A\|_{\infty}\|B\|_{\infty} + |\beta|\|C_{\text{init}}\|_{\infty})\varepsilon} \leq 1
\]
Backward error analysis of TRSM  [Nicholas J. Higham, 2002]

**Theorem:** Let the triangular systems $Tx = b$ where $T \in \mathbb{R}^{n \times n}$ is nonsingular, be solved by substitution with any ordering. Then the computed solution $\hat{x}$ satisfies $(T + \Delta T)\hat{x} = b$, $|\Delta T| \leq \gamma_n |T|$, where $\gamma_n = \frac{n \epsilon}{1 - n \epsilon} \approx n \epsilon$.

Backward error bound of TRSM

- **Formula:** $AX = \alpha B$
- **Backward error bound:** $N \|A\|_{\infty} \|X\|_{\infty} \epsilon$
- **Note:** TRSM’s backward error bound does depend on $B$
- **The solution** $\hat{X}$ is acceptable if $\frac{\|A\hat{X} - \alpha B\|_{\infty}}{N \|A\|_{\infty} \|\hat{X}\|_{\infty} \epsilon} \leq 1$
### Accuracy testing: BLAS3 error bound summary

<table>
<thead>
<tr>
<th>Routine</th>
<th>Formula</th>
<th>Error bound based success criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEMM</td>
<td>$C = \alpha AB + \beta C_{init}$</td>
<td>$\frac{|C - \hat{C}|_{\infty}}{(N</td>
</tr>
<tr>
<td>SYMM</td>
<td>$C = \alpha A^T H + \beta C_{init}$</td>
<td>$\frac{|C - \hat{C}|_{\infty}}{(N</td>
</tr>
<tr>
<td>HEMM</td>
<td>$C = \alpha AB^T H + \alpha B A^T H + \beta C_{init}$</td>
<td>$\frac{|C - \hat{C}|_{\infty}}{(N</td>
</tr>
<tr>
<td>SYR2K</td>
<td>$AX = \alpha B$</td>
<td>$\frac{|AX - \alpha B|_{\infty}}{N</td>
</tr>
</tbody>
</table>
Accuracy testing: some remarks on the forward error bound

\[ \frac{\| C - \hat{C} \|_\infty}{(N|\alpha|\|A\|_\infty\|B\|_\infty + |\beta|\|C_{init}\|_\infty)\epsilon} \leq 1? \]

In our accuracy checking, we assume that the correct solution \( C \) of GEMM exists, what is not true in real-world applications.

\[ \frac{\| C - \hat{C} \|_\infty}{(N|\alpha|\|A\|_\infty\|B\|_\infty + |\beta|\|C_{init}\|_\infty)\epsilon} \leq 2? \]

- Our exact solutions: reference BLAS
- Problem: reference BLAS results have the same error bound
- Pessimistic approach: authorise twice the error bound
Outline

1. Reference implementation
2. Accuracy testing
3. Experimental results
4. Concluding remarks
Some implementation details

Batched BLAS implementations tested

- **Batched_cUBLAS**: GEMM and TRSM
- **Batched_MAGMA**: GEMM, HERK, SYRK and TRSM
- **Batched_MKL**: only GEMM

Alternative to Batched on multi-core processor?

- **OMP_LOOP_MKL**: OpenMP loop over sequential MKL
- **PARALLEL_MKL**: single loop over parallel MKL
About the source code

- Programming language: C
- Precision: single, double, single complex and double complex
- Documentation: well documented (doxygen)
- First release coming soon

Hardware

- CPU: 2x10 Intel Xeon E5-2650 v3, 25M Cache, 2.30 GHz
- GPU: Nvidia K40c GPU with 2, 880 CUDA cores
The University of Manchester - Reference implementation and testing

Experimental results

Accuracy of 1K batched DGEMM

Forward error bound

Limit = 2

Matrix size $M = K = N$

Batched_MAGMA
Batched_cuBLAS
Experimental results

Accuracy of 1K batched DTRSM

Backward error bound vs Matrix size M = N

Limit = 1

Batched_MAGMA
Batched_cuBLAS
Performance of 1K batched DGEMM

Matrix size $M = K = N$

- Batched_MAGMA
- Batched_cuBLAS
- Batched_MKL
- OMP_LOOP_MKL
- PARALLEL_MKL

Gflop/s vs Matrix size $M = K = N$
Performance of 1K batched DGEMM

![Graph showing performance comparison between Batched_MAGMA, Batched_cuBLAS, Batched_MKL, OMP_LOOP_MKL, and PARALLEL_MKL for various matrix sizes M = K = N. The graph plots Gflop/s against matrix size with a logarithmic scale for better visualization. The performance improves as the matrix size increases for all implementations.](image-url)
Outline

1. Reference implementation
2. Accuracy testing
3. Experimental results
4. Concluding remarks
Concluding remarks and perspectives

**Conclusion remarks**

- Investigation of new accuracy checking
- Competitive performance cuBLAS/MAGMA
- MKL batched BLAS seems promising

**Future work**

- First release with respect to the final specification
- Extension to level 1 & 2 BLAS
- Provide fortran interface
**Conclusion remarks**

- Investigation of new accuracy checking
- Competitive performance cuBLAS/MAGMA
- MKL batched BLAS seems promising

**Future work**

- First release with respect to the final specification
- Extension to level 1 & 2 BLAS
- Provide fortran interface