Reproducible, Accurately Rounded and Efficient (RARE) BLAS

Chemseddine Chohra
Philippe Langlois et David Parello

University of Perpignan Via Domitia

RAIM - June 29th 2016
Limited Machine Precision

- Using floating point numbers as approximation.
- \( x \longrightarrow X = \text{fl}(x) \) if \( x \notin F \) or \( x \) if \( x \in F \).
- \( X + Y \neq X \oplus Y = \text{fl}(X + Y) \).
- IEEE-754 standard defines several rounding modes.

Faithfully rounded

\[ \sum X_i \]

Faithfully rounded

\[ \sum X_i \]

RTN
Non-associativity of Addition

- \( A \oplus (B \oplus C) \neq (A \oplus B) \oplus C. \)
- Catastrophic cancelation: \( M = 2^{53}; \quad 0 = -M \oplus (M \oplus 1) \neq (-M \oplus M) \oplus 1 = 1. \)

Non-reproducibility of Summation

- For a sum \( (\sum_{i=1}^{n} X_i) \), the final result depends on the order of the computations.
- Why could operations order be different?
  - Dynamic data scheduling.
  - Non-deterministic reductions.
  - Resources availability.
  - Different instruction sets.
Is Numerical Reproducibility Really Important?

- Important for debugging.
- Important for validating results.
- Reproducibility: One of top 10 exascale research challenges (U.S. Department of Energy [DOE], 2014).
  - $10^{18}$ flop/s.
  - Millions of cores.

"Reproducibility will be expensive if not impossible to achieve on exascale"
Introduction and Problematic

Parallel Libraries Solutions

### Solutions for Reproducibility Problem
- Static scheduling (OpenMP).
- Deterministic reduction.
- Intel MKL: CNR.

### Algorithmic Solutions
- Deterministic error.
  - ReprodSum (Demmel and Nguyen, 2013).
  - FastReprodSum (Demmel and Nguyen, 2013).
  - OneReduction (Demmel and Nguyen, 2014).
  - ReprodBLAS library.
- Higher precision (quadruple precision for instance).
  - SumK and DotK (Ogita and al., 2005).
  - Improve accuracy and consistency (Villa and al., 2009).
  - Reproducibility is not always guaranteed.
- Correctly rounded arithmetic.
  - FP expansions + Super accumulators (Collange and al., 2014).
  - Small and Large Super accumulators (Neal, 2015).
Our Aim

Guarantee the Numerical Reproducibility for BLAS (Basic Linear Algebra Subroutines)

- **Level 1**: `_amax`, `_swap`, `_copy`, `_scal`, `_axpy`, `_nrm2`, `_asum`, `_dot`.
- **Level 2**: `_gemv`, `_trsv`, `_ger`, `_syr`, `_syr2`.
- **Level 3**: `_gemm`, `_syrk`, `_syrk2`, `_trsm`.

In This Talk

- Shared memory parallel implementation.
- Distributed memory parallel implementation.
- Xeon Phi implementation.
Our Aim

Guarantee the Numerical Reproducibility for BLAS (Basic Linear Algebra Subroutines)

- Level 1: i_amax, _swap, _copy, _scal, _axpy, _nrm2, _asum, _dot.
- Level 2: _gemv, _trsv, _ger, _syr, _syr2.
- Level 3: _gemm, _syrk, _syrk2, _trsm.

In This Talk

- Shared memory parallel implementation.
- Distributed memory parallel implementation.
- Xeon Phi implementation.
Our Aim

Guarantee the Numerical Reproducibility for BLAS (Basic Linear Algebra Subroutines)

- Level 1: `i_amax`, `_swap`, `_copy`, `_scal`, `_axpy`, `_nrm2`, `_asum`, `_dot`.
- Level 2: `_gemv`, `_trsv`, `_ger`, `_syr`, `_syr2`.
- Level 3: `_gemm`, `_syrk`, `_syrk2`, `_trsm`.

In This Talk

- Shared memory parallel implementation.
- Distributed memory parallel implementation.
- Xeon Phi implementation.
Table of Contents

1 Introduction and Problematic

2 RARE-BLAS

3 Performance and Accuracy Results

4 Conclusion and Future Work
1 Introduction and Problematic

2 RARE-BLAS
   - Dot Product
   - Euclidean Norm
   - Sum of Absolute Values
   - Matrix Vector Multiplication

3 Performance and Accuracy Results

4 Conclusion and Future Work
Parallel Dot Product

\[ \sum_{i} (X_i \times Y_i) = \sum_{j} P_j \]

\[ \text{Size}(T) \leq \frac{\text{ExponentRange}}{\text{SignificandSize}} \]

\[ \text{Size}(T) \leq 40 \text{ for binary64} \]
Parallel Dot Product

\[
\sum_{i} (X_i \times Y_i) = \sum_{j} P_j \\
\text{Size}(T) \leq \frac{\text{ExponentRange}}{\text{SignificandSize}} \\
\text{Size}(T) \leq 40 \text{ for binary64}
\]
Parallel Dot Product

\[ \sum_{i} (X_i \times Y_i) = \sum_{j} P_j \]
Parallel Dot Product

\[
\begin{align*}
X[n] & \quad Y[n] \\
\vdots & \quad \vdots \\
\vdots & \quad \vdots \\
\vdots & \quad \vdots \\
\vdots & \quad \vdots \\
\end{align*}
\]

\[
\text{Error-Free Transformation}
\]

\[
T
\]

\[
\begin{align*}
\sum_i (X_i \times Y_i) &= \sum_j P_j \\
\end{align*}
\]

\[
iFastSum(P) = RTN(\sum_i (X_i \times Y_i))
\]

Size(\(T\)) ≤ ExponentRange / SignificandSize

Size(\(T\)) ≤ 40 for binary64

Chemseddine Chohra (UPVD)

RARE-BLAS

Dot Product

Reproducible BLAS

June 29th 2016 9 / 28
Parallel Dot Product

\[ X[n] \cdot Y[n] \]

Error-Free Transformation

\[ T \]

\[ \text{Size}(T) \leq \frac{\text{ExponantRange}}{\text{SignificandSize}} \]

\[ iFastSum(P) = RTN(\sum_i (X_i \times Y_i)) \]
Parallel Dot Product

\[ \sum_{i} (X_i \times Y_i) = \sum_{j} P_j \]

\[ iFastSum(P) = RTN(\sum_{i}(X_i \times Y_i)) \]

\[ \text{Size}(T) \leq 40 \text{ for binary64} \]
Parallel Dot Product

\[ X[n] \cdot Y[n] \]

\[ \text{Error-Free Transformation} \]

\[ T \]

\[ \text{Size}(T) \leq 40 \text{ for binary64} \]

\[ \text{iFastSum}(P) = RTN(\sum_i (X_i \times Y_i)) \]
About Algorithms

- Accumulate together the elements with the same exponent.
- Extra precision is simulated in two ways.
  - Split the input so the standard numbers are considered as accumulators.
  - Use quadruple precision.

...
About Algorithms

- Accumulate together the elements with the same exponent.
- Extra precision is simulated in two ways.
  - Split the input so the standard numbers are considered as accumulators.
  - Use quadruple precision.
About Algorithms

- Accumulate together the elements with the same exponent.
- Extra precision is simulated in two ways.
  - Split the input so the standard numbers are considered as accumulators.
  - Use quadruple precision.

\[ \begin{array}{cccccc}
+ & & & & \\
+ & & & & \\
\ldots & \ldots & \ldots & \ldots & \\
+ & & & & \\
\hline
\end{array} \]

\[ \begin{array}{cccccc}
+ & & & & \\
+ & & & & \\
\ldots & \ldots & \ldots & \ldots & \\
+ & & & & \\
\hline
\end{array} \]

\[ \begin{array}{cccccc}
= & & & & \\
+ & & & & \\
\ldots & \ldots & \ldots & \ldots & \\
+ & & & & \\
\hline
\end{array} \]

\[ \begin{array}{cccccc}
= & & & & \\
+ & & n & & \\
\ldots & \ldots & \ldots & \ldots & \\
+ & & & & \\
\hline
\end{array} \]

\[ 2^n \text{ summands} \]
Algorithm OnlineExact (Zhu and Hayes, 2010)

\[ X[n] \]

Accumulate elements with the same exponent

\[ \sum_{i=1}^{n} X_i = \sum_{j=1}^{2048} C_j \]
Algorithm OnlineExact (Zhu and Hayes, 2010)

\[ \sum_{i=1}^{n} X_i = \sum_{j=1}^{2048} C_j \]

Accumulate elements with the same exponent

\[ iFastSum(C) = RTN(\sum_{i=1}^{n} X_i) \]
Efficiency of the Algorithm OnlineExact

Implementation

- Entry vector condition number = $10^{16}$.

Note

- The transformation cost is linear to vector size.
- Post-transformation process cost is negligible for large vectors.
- It is better to use an iterative algorithm on small datasets.
Error-Free Transformation

\[ \sum_{i=1}^{n} X_i \times Y_i \]

\[ \text{result} \]

\[ \text{error} \]

\[ \text{iFastSum}(C) = RTN(\sum_{i=1}^{n} X_i \times Y_i) \]
Error-Free Transformation

\[ \begin{align*}
X[n] & \quad \text{TwoProd}(X_i, Y_i) \\
Y[n] & \quad \text{error} \\
\text{result} & \quad \text{error} \\
C[2048] & \end{align*} \]

result error

algorithm

\[ \text{iFastSum}(C) = \text{RTN}(\sum_{i=1}^{n} X_i \times Y_i) \]

Distillation

Chemseddine Chohra (UPVD)  Reproducible BLAS  June 29th 2016
Error-Free Transformation

\[ \text{TwoProd}(X_i, Y_i) \rightarrow \text{Distillation} \]

result

c\[2048\]

error

\( T \)
Euclidean Norm

How to Get a Reproducible Euclidean Norm

- $Nrm2(X) = \sqrt{\sum_{i=1}^{n} X_i^2}$.
- If $\sum_{i=1}^{n} X_i^2$ is correctly rounded, $\sqrt{\sum_{i=1}^{n} X_i^2}$ is faithfully rounded (Graillat and al., 2014).
- $Dot(X,X) = \sum_{i=1}^{n} X_i^2$ is correctly rounded.
- $Nrm2(X) = \sqrt{Dot(X,X)}$ is faithfully rounded and reproducible.
How to Get a Faithfully Rounded Sum of Absolute Values

- The condition number is always 1.
- Error bounds depend only on size of vector.
- Algorithm SumK can be used.
- $K$ is defined according to vector size.
Matrix Vector Multiplication

\[ y = \alpha \cdot A \cdot x + \beta \cdot y \]

Algorithm Steps

1. \[ y_i = \alpha \cdot (a_i \cdot x) + \beta \cdot y_i \]
2. \[ \text{EFT} \left( a_i \cdot x \right) = T \Rightarrow \sum_j (a_i \cdot x_j) = \sum_k T_k \]
3. \[ y_i = \alpha \cdot \left( \sum_k T_k \right) + \beta \cdot y_i \]
4. \[ \text{RTN} \left( \sum_k (\alpha \cdot T_k) + \beta \cdot y_i \right) \]
Matrix Vector Multiplication

\[ y = \alpha \cdot A \cdot x + \beta \cdot y \]

\[ y_i = \alpha \cdot a^{(i)} \cdot x + \beta \cdot y_i \]

\[ \sum_j (a^{(i)}_j \cdot x_j) = \sum_k T_k \cdot y_i \]

\[ y_i = \alpha \cdot (\sum_k T_k) + \beta \cdot y_i \]

\[ y_i = \text{RTN} (\sum_k (\alpha \cdot T_k) + \beta \cdot y_i) \]
Matrix Vector Multiplication

Algorithm Steps

- \( y_i = \alpha \cdot (a^{(i)} \cdot x) + \beta \cdot y_i \)
- \( EFT(a^{(i)} \cdot x) = T \Rightarrow \sum_j (a_j^{(i)} \cdot x_j) = \sum_k T_k \)
- \( y_i = \alpha \cdot (\sum_k T_k) + \beta \cdot y_i \)
Matrix Vector Multiplication

Algorithm Steps

- $y_i = \alpha \cdot (a^{(i)} \cdot x) + \beta \cdot y_i$
- $EFT(a^{(i)} \cdot x) = T \Rightarrow \sum_j (a_j^{(i)} \cdot x_j) = \sum_k T_k$
- $y_i = \alpha \cdot (\sum_k T_k) + \beta \cdot y_i$
- $y_i = \text{RTN}(\sum_k (\alpha \cdot T_k) + \beta \cdot y_i)$
Table of Contents

1 Introduction and Problematic

2 RARE-BLAS

3 Performance and Accuracy Results
   • Experimental Frameworks
   • Dot Product
   • Euclidean Norm
   • Sum of Absolute Values
   • Matrix Vector Multiplication
   • Accuracy Results

4 Conclusion and Future Work
Experimental Framework

Shared Memory
- dual Xeon E5-2650 v2 16 cores (8 per socket).

Distributed Memory
- OCCIGEN (26th supercomputer in top500 list).
- 4212 Xeon E5-2690 v3 socket (12 cores per socket).

Accelerator
- Intel Xeon Phi 7120 accelerator, 60 cores.
Performance and Accuracy Results

Results for Dot Product

Implementation

- Manually optimized version for all algorithms.
- Entry vectors condition number = $10^8$.

Shared Memory Performance

Xeon Phi Performance
Dot Product

Results for Dot Product

Configurations

- \#sockets = 1 .. 128.
- \#threads = 12 per socket.

Dataset

- Entry vectors size = $10^7$.
- Condition number = $10^{32}$.

Note

- Good scaling for large datasets.
- Two communications cost limits ReprodDot and FastReprodDot.
- We need only one communication for OneReduction, HybridSum and OnlineExact.
Results for Euclidean Norm

Implementation

- Manually optimized version for all algorithms.

Shared Memory Performance

Xeon Phi Performance
Implementation

- Manually optimized version for all algorithms.

Shared Memory Performance

Xeon Phi Performance
Results for Matrix Vector Multiplication

Implementation

- Manually optimized version for all algorithms.
- Entry condition number = \(10^8\).

Shared Memory Performance

Xeon Phi Performance
Accuracy Results

Accuracy of Dot Product (size = $10^5$)  

Accuracy of Gemv ($m = n = 1000$)
Table of Contents

1 Introduction and Problematic
2 RARE-BLAS
3 Performance and Accuracy Results
4 Conclusion and Future Work
Conclusion

Guarantee the Numerical Reproducibility for BLAS (Basic Linear Algebra Subroutines)

- Level 1: \texttt{i\_amax, \_swap, \_copy, \_scal, \_axpy, \_nrm2, \_asum, \_dot}.
- Level 2: \texttt{\_gemv, \_trsv, \_ger, \_syr, \_syr2}.
- Level 3: \texttt{\_gemm, \_syrk, \_syrk2, \_trsm}.

Reproducible Level 1 BLAS

- RTN cost for BLAS is acceptable on CPUs ($1 \times -2\times$).
- Xeon Phi performance are lower but still useful for debugging and validation ($2 \times -6\times$).
- Only one pass through the vector and one communication are required.
- Our solution do not depend on condition.
Conclusion and Future Work

Conclusion

Guarantee the Numerical Reproducibility for BLAS (Basic Linear Algebra Subroutines)

- Level 1: i_amax, _swap, _copy, _scal, _axpy, _nrm2, _asum, _dot.
- Level 2: _gemv, _trsv, _ger, _syr, _syr2.
- Level 3: _gemm, _syrk, _syrk2, _trsm.

Reproducible Level 1 BLAS

- RTN cost for BLAS is acceptable on CPUs ($1 \times -2\times$).
- Xeon Phi performance are lower but still useful for debugging and validation ($2 \times -6\times$).
- Only one pass through the vector and one communication are required.
- Our solution do not depend on condition.
Future Work

Work in Progress
- Reproducible Triangular Solver.

Future Work
- Level 2 BLAS : _ger.
- Level 3 BLAS : _gemm, _trsm.
- Matrix decompositions.
THANK YOU FOR YOUR ATTENTION
About Algorithms

- Inspired from AccSum (Rump and al, 2008) and FastAccSum (Rump and al, 2009).

Steps for Sequential

- Compute max.

\[ X_1 \]
\[ X_2 \]
\[ X_3 \]
\[ \vdots \]
\[ \vdots \]
\[ X_n \]
About Algorithms

- Inspired from AccSum (Rump and al, 2008) and FastAccSum (Rump and al, 2009).

Steps for Sequential

- Compute max.

\[ \max \]

\[
\begin{align*}
X_1 \\
\vdots \\
X_n \\
\sigma
\end{align*}
\]

E_{\text{max}} \quad E_{\text{min}}
About Algorithms

- Inspired from AccSum (Rump and al, 2008) and FastAccSum (Rump and al, 2009).

Steps for Sequential

- Compute max.
About Algorithms

- Inspired from AccSum (Rump and al, 2008) and FastAccSum (Rump and al, 2009).

Steps for Sequential

- Compute max.
- Sum each fold.
ReproSum and FastReproSum (Demmel and Nguyen, 2013)

About Algorithms

- Inspired from AccSum (Rump and al, 2008) and FastAccSum (Rump and al, 2009).

Emin

Emax

K = 1  K = 2  K = 3

Steps for Sequential

- Compute max.
- Sum each fold.
About Algorithms

- Inspired from AccSum (Rump and al, 2008) and FastAccSum (Rump and al, 2009).

Steps for Sequential
- Compute max.
- Sum each fold.

Steps for Parallel
- Max (Compute).
- Max (Reduce).
- Sum (Compute).
- Sum (Reduce).
OneReduction (Demmel and Nguyen, 2013)

Steps for Parallel

- $X_1$
- $X_2$
- $X_3$
- ...
- $X_n$
OneReduction (Demmel and Nguyen, 2013)

Steps for Parallel

$X_1$
$X_2$
$X_3$
\vdots
\vdots
$X_n$
OneReduction (Demmel and Nguyen, 2013)

Steps for Parallel

Thread 0

Thread 1

Thread 2
OneReduction (Demmel and Nguyen, 2013)

Steps for Parallel

- Max (Compute).

{E_{max}} \quad {X_1} \quad {E_{min}}

{X_2}

{X_3}

\ldots

{X_n}

Thread 0

Thread 1

Thread 2
OneReduction (Demmel and Nguyen, 2013)

Steps for Parallel
- Max (Compute).
- Sum (Compute).
OneReduction (Demmel and Nguyen, 2013)

Steps for Parallel
- Max (Compute).
- Sum (Compute).
OneReduction (Demmel and Nguyen, 2013)

Steps for Parallel
- Max (Compute).
- Sum (Compute).

Emax

Emin

X_1

X_2

X_3

X_n

Thread 0

Thread 1

Thread 2
OneReduction (Demmel and Nguyen, 2013)

Steps for Parallel
- Max (Compute).
- Sum (Compute).
- Sum (Reduce).
OneReduction (Demmel and Nguyen, 2013)

**Pros**
- Always reproducible result.
- Easy to enhance precision.
- Single communication.

**Cons**
- Accuracy problem on ill-conditioned sums.
Algorithm HybridSum (Zhu and Hayes, 2009)

\[
\sum_{i=1}^{n} X_i = \sum_{i=1}^{2048} C_i
\]
Algorithm HybridSum (Zhu and Hayes, 2009)

\[ iFastSum(C) = RTN(\sum_{i=1}^{n} X_i) \]
Algorithm OnlineExact (Zhu and Hayes, 2010)

\[
X[n]
\]

\[
\text{FastTwoSum}(X_i, C_1 \exp(X_i))
\]

\[
C_2 \exp(X_i) + \text{Error}
\]

\[
i\text{FastSum}(C_1 \cup C_2) = \text{RTN}(\sum_{i=1}^{n} X_i)
\]

Chemseddine Chohra (UPVD)
Implementation

- Entry vector condition number = $10^{16}$.

Note

- The transformation cost is linear to vector size.
- Post-transformation process cost is negligible for large vectors.
- It is better to use an iterative algorithm on small datasets.