Reproducible level 1 BLAS on massively parallel systems

Chemseddine Chohra
Philippe Langlois and David Parello

University of Perpignan Via Domitia

RAIM - April 07th 2015
Limited machine precision

- Using floating point numbers as approximation.
- \( x \rightarrow X = \text{fl}(x) \) 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.
Non-associativity of addition

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

Non-reproducibility of summation

- For a sum $(\sum_{i=1}^{n} X_i)$, the final result depends on the order of the computations.
- Why should operations order be different.
  - Different micro-architectures (different instruction sets).
  - Different compilers (or even versions of compilers).
  - Data alignment.
  - Dynamic scheduling.
  - Non-deterministic reduction.
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 problematics

How to fix the numerical reproducibility problem?

Parallel libraries solutions

- Static scheduling.
- Deterministic reduction.
  - Available on OpenMP and TBB.
  - Recommended in MPI 3.0 standard.
- MKL (Conditional Numerical Reproducibility).

Algorithmic solutions

- Deterministic error (Demmel and Nguyen, 2013).
  - ReproSum.
  - FastReproSum.
  - OneReduction.
- Enhanced precision.
  - Higher precision (quadruple precision for instance).
    - Improve accuracy and consistency (Villa and al., 2009).
    - Reproducibility is not always guaranteed.
  - Correctly rounded arithmetic.
    - FP expansions + Super accumulators (S. Collange and al., 2014).
Our aim

Guarantee the numerical reproducibility for BLAS (Basic Linear Algebra Subroutines)

- Level 1: max, min, scal, axpy, norm, asum, dot.
- dot can be transformed to a sum $\sum_{i=1}^{n} X_i \cdot Y_i = \sum_{i=1}^{2n} Z_i$.

Compute an accurate sum

- Several algorithms available.
- Is the cost acceptable?
1 Introduction and problematic

2 Algorithms for reproducibility

3 Precise summation algorithms

4 Cost of parallel implementation

5 Conclusion
Reproducible level 1 BLAS

Chemseddine Chohra (UPVD)

Algorithms for reproducibility

ReprodSum and FastReprodSum (Demmel and Nguyen, 2013)

$X_1$

$X_2$

$X_3$

\vdots

$X_n$

Steps for sequential

- Compute max.
Steps for sequential

- Compute max.
RepordSum and FastRepordSum (Demmel and Nguyen, 2013)

Steps for sequential

- Compute max.

---

Max (Compute).
Max (Reduce).
Sum (Compute).
Sum (Reduce).

---

E_{max}

\sigma

\begin{align*}
X_1 & \quad \text{---} \\
X_2 & \quad \text{---} \\
X_3 & \quad \text{---} \\
\vdots & \quad \text{---} \\
X_n & \quad \text{---}
\end{align*}
ReproSum and FastReproSum (Demmel and Nguyen, 2013)

Steps for sequential
- Compute max.
- Sum each fold.
Reproducible level 1 BLAS

Chemseddine Chohra (UPVD)
April 07th 2015

ReproSum and FastReproSum (Demmel and Nguyen, 2013)

Steps for sequential:
- Compute max.
- Sum each fold.

Diagram:
- Emax
- Emin
- X1
- X2
- X3
- ... (ellipses)
- Xn
- σ
- Thread 0
- Thread 1
- Thread 2
- Compute max.
- Sum each fold.
- Max (Compute)
- Max (Reduce)
- Sum (Compute)
- Sum (Reduce)
Reproducible level 1 BLAS

Algorithms for reproducibility

ReprodSum and FastReprodSum (Demmel and Nguyen, 2013)

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

Steps for parallel
- Max (Compute).
- Max (Reduce).
- Sum (Compute).
- Sum (Reduce).

$X_1 \ldots X_n \sigma$

$E_{\text{max}}$ $E_{\text{min}}$

Thread 0

Thread 1

Thread 2
Algorithms for reproducibility

OneReduction (Demmel and Nguyen, 2013)

Steps for parallel

$E_{\text{max}}$  $E_{\text{min}}$

$X_1$

$X_2$

$X_3$

$\vdots$

$X_n$
Algorithms for reproducibility

OneReduction (Demmel and Nguyen, 2013)

Steps for parallel

$\mathbf{x}_1$

$\mathbf{x}_2$

$\mathbf{x}_3$

$\vdots$

$\vdots$

$\mathbf{x}_n$
OneReduction (Demmel and Nguyen, 2013)

Steps for parallel

- Max (Compute)
- Sum (Compute)
- Sum (Reduce)
OneReduction (Demmel and Nguyen, 2013)

Algorithms for reproducibility

Steps for parallel

- Max (Compute).

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

\begin{align*}
X_1 & \quad \text{Thread 0} \\
X_2 & \quad \text{Thread 1} \\
X_3 & \quad \text{Thread 2} \\
\vdots & \\
X_n &
\end{align*}
Algorithms for reproducibility

OneReduction (Demmel and Nguyen, 2013)

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

Chemseddine Chohra (UPVD)
Reproducible level 1 BLAS
April 07th 2015
OneReduction (Demmel and Nguyen, 2013)

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

E_{max} \quad E_{min}

X_1

X_2

X_3

\vdots

X_n

Thread 0

Thread 1

Thread 2
Algorithms for reproducibility

OneReduction (Demmel and Nguyen, 2013)

Steps for parallel

- Max (Compute).
- Sum (Compute).

E_{max}

E_{min}

\begin{align*}
X_1 & \quad \text{Thread 0} \\
X_2 & \quad \text{Thread 0} \\
X_3 & \quad \text{Thread 1} \\
\vdots & \quad \text{Thread 1} \\
X_n & \quad \text{Thread 2}
\end{align*}
OneReduction (Demmel and Nguyen, 2013)

Steps for parallel:
- Max (Compute).
- Sum (Compute).
- Sum (Reduce).

Diagram:
- Variables: $X_1, X_2, X_3, \ldots, X_n$
- Threads: Thread 0, Thread 1, Thread 2
- Steps:
  - Max (Compute)
  - Sum (Compute)
  - Sum (Reduce)
1 Introduction and problematic
2 Algorithms for reproducibility
3 Precise summation algorithms
4 Cost of parallel implementation
5 Conclusion
Precise summation algorithms

**Faithfully rounded (one of the floating-point neighbors)**
- FastSum (Zhu and Hayes, 2005).
- AccSum (Rump and al., 2008).
- FastAccSum (Rump, 2008).

**Correctly rounded (according to the rounding mode)**
- NearSum (Rump and al., 2008).
- iFastSum (Zhu and Hayes, 2009).
- HybridSum (Zhu and Hayes, 2009).
- OnlineExact (Zhu and Hayes, 2010).
Precise summation algorithms

Experimental framework

Implementation

- Implemented using C language.

Compiler

- Intel ICC 14.0.2.
- Options: -O3 -xHost -fp-model double -fp-model strict -funroll-all-loops.

Hardware

- i7-3540M at 3 GHz.
- Turbo boost turned off.
HybridSum and OnlineExact do not depend on the condition number

**Implementation**

- Manually optimized version for all algorithms.
- Entry vectors size $= 10^6$.

**Note**

- HS and OLE: condition number independents.
Precise summation algorithms

Algorithm OnlineExact (Zhu and Hayes, 2010)

\[
X[n] \rightarrow \text{FastTwoSum}(X_i, C_1^{\exp}(X_i)) \rightarrow C_2^{\exp}(X_i) \rightarrow C_1[2048] \rightarrow C_2[2048] \rightarrow i\text{FastSum}(C_1 \cup C_2)
\]

\[X[n] = \ldots \]

\[C_1 = \ldots \]

\[C_2 = \ldots \]
Precise summation algorithms

Dot with OnlineExact (Zhu and Hayes, 2010)

TwoProd($X_i, Y_i$)
Precise summation algorithms

HybridSum and OnlineExact are not efficient for small vectors

Implementation

- Manually optimized version for all algorithms.
- Entry vectors condition = $10^{16}$.

Note

- HS and OLE: more efficient for large vectors.
- Mixed solution is possible.
1 Introduction and problematic
2 Algorithms for reproducibility
3 Precise summation algorithms
4 Cost of parallel implementation
   - Parallel RTN algorithm
   - Experimental framework
   - Experimental results
5 Conclusion
Parallel OnlineExact (2 processors case)
Parallel OnlineExact (2 processors case)
Parallel OnlineExact (2 processors case)
Experimental framework

Hardware
- OCCIGEN (26th supercomputer in top500 list).
- 4212 Xeon E5-2690 v3 socket (L3 cache = 30 M).
- 12 cores on each socket.

Software
- Intel ICC 15.0.0.
- OpenMP 4.0 (Intra socket parallelism).
- OpenMPI (Inter socket communications).

Algorithms
- Classic dot (MKL implementation + MPI reduction).
- Reproducible dot (ReprodDot, FastReprodDot, OneReduction).
- Correctly rounded dot (Parallel version of OnlineExact and HybridSum).
Single socket results

**configurations**
- #sockets = 1.
- #threads = 1 .. 12 on single socket.

**dataset**
- Entry vectors size = $10^6$.
- Condition number = $10^{32}$.

**Note**
- Classic dot is memory bounded.
- Data hold in level 3 cache.
- ReprodDot and FastReprodDot avoid second memory access.
Single socket results

configurations
- \#sockets = 1.
- \#threads = 1 .. 12 on single socket.

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

Note
- Data do not hold in cache.
- Memory cost is more important for ReprodDot and FastReprodDot.
Multi-socket results

- **configurations**
  - \#sockets = 1 .. 128.
  - \#threads = 8 per socket.

- **dataset**
  - Entry vectors size = $10^6$.
  - Condition number = $10^{32}$.

- **Note**
  - Small dataset.
  - We do not need too much sockets.
Multi-socket results

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

**dataset**
- Entry vectors size = $10^6$.
- Condition number = $10^{32}$.

**Note**
- Small dataset.
- We do not need too much sockets.
## Cost of parallel implementation

### Multi-socket results

#### Configurations
- \#sockets = 1 .. 128.
- \#threads = 8 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.

### Experimental results

<table>
<thead>
<tr>
<th>Configurations</th>
<th>dataset</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>#sockets = 1 .. 128.</td>
<td>Entry vectors size = $10^7$.</td>
<td>Good scaling for large datasets.</td>
</tr>
<tr>
<td>#threads = 8 per socket.</td>
<td>Condition number = $10^{32}$.</td>
<td>Two communications cost limits ReprodDot and FastReprodDot.</td>
</tr>
</tbody>
</table>

### Diagram

- ClassicDot
- OneReductionDot
- FastReprodDot
- ReprodDot
- OnlineExactDot
- HybridSumDot

<table>
<thead>
<tr>
<th>Sockets</th>
<th>Runtime (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5e+07</td>
</tr>
<tr>
<td>2</td>
<td>3.0e+07</td>
</tr>
<tr>
<td>4</td>
<td>2.5e+07</td>
</tr>
<tr>
<td>8</td>
<td>2.0e+07</td>
</tr>
<tr>
<td>16</td>
<td>1.5e+07</td>
</tr>
<tr>
<td>32</td>
<td>1.0e+07</td>
</tr>
<tr>
<td>64</td>
<td>5.0e+06</td>
</tr>
<tr>
<td>128</td>
<td>0</td>
</tr>
</tbody>
</table>
Multi-socket results

**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.
- Using all cores: advantageous for RTN implementations.
Table of contents

1 Introduction and problematic
2 Algorithms for reproducibility
3 Precise summation algorithms
4 Cost of parallel implementation
5 Conclusion
Conclusion

Reproducible level 1 BLAS

- RTN cost for level 1 BLAS is acceptable on massively multi-thread systems.
- Our solution do not depend on condition.
- Intra-socket performance for classic dot is memory bounded.
- Only one communication is required.
- Scale correctly up to 128 sockets.
Future work

- Level 2 BLAS.
  - GEMV.
  - TRSV.

- Level 3 BLAS will not be easy.
  - DGEMM utilise 95% of peak performance.
  - Used algorithms are not adapted to work on blocks.
  - Some operations can not be vectorised.
  - No balance between FP additions and multiplications.
THANK YOU FOR YOUR ATTENTION