Reproducible, Accurately Rounded and Efficient (RARE) BLAS

Chemseddine Chohra
Philippe Langlois et David Parello

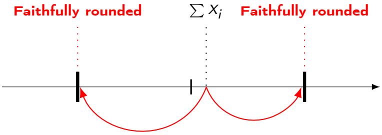
University of Perpignan Via Domitia

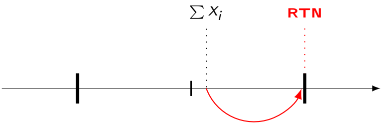
RAIM - June 29^{th} 2016

Introduction and Problematic

Limited Machine Precision

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





Introduction and Problematic

Non-associativity of Addition

- A ⊕ (B ⊕ C) ≠ (A ⊕ B) ⊕ C.
- Catastrophic cancelation : M = 2^{53}; 0 = -M ⊕ (M ⊕ 1) ≠ (-M ⊕ M) ⊕ 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.

Introduction and Problematic

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"

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 : 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.

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.

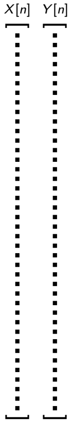
In This Talk

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

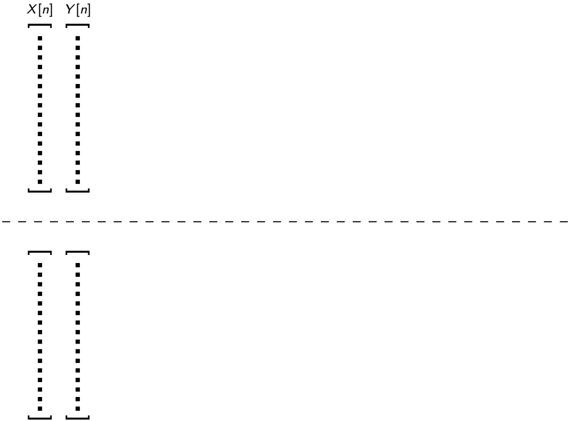
Table of Contents

Table of Contents

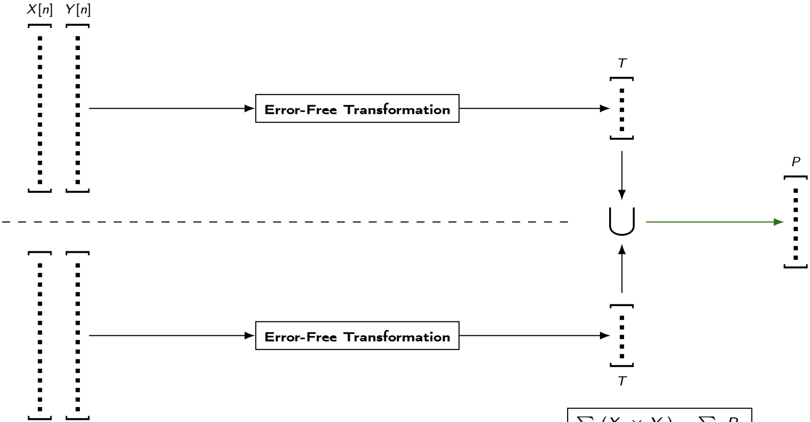
Parallel Dot Product



Parallel Dot Product

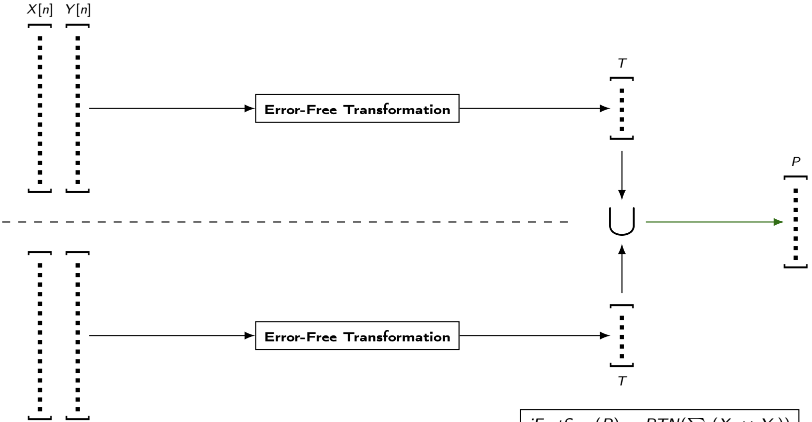


Parallel Dot Product



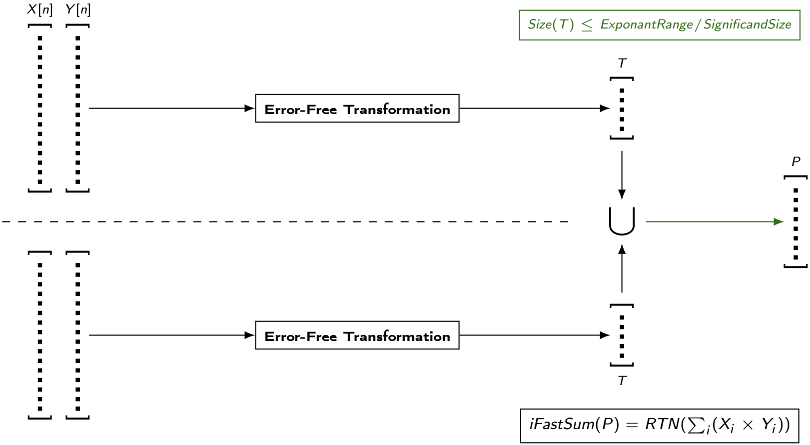
$\sum_i (X_i \times Y_i) = \sum_j P_j$

Parallel Dot Product

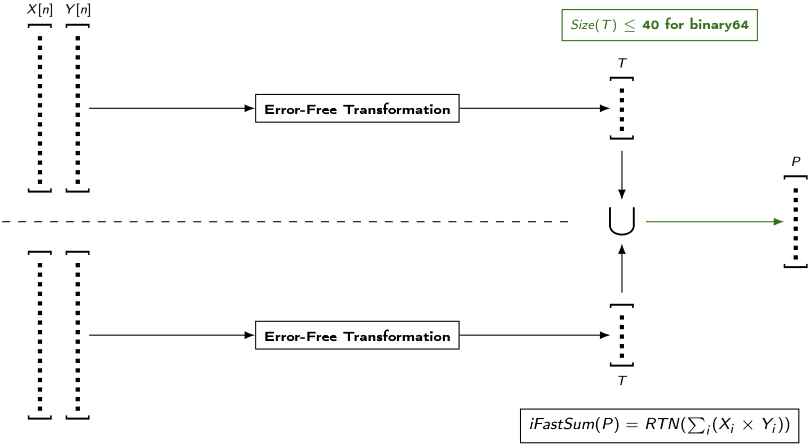


$iFastSum(P) = RTN(\sum_i (X_i \times Y_i))$

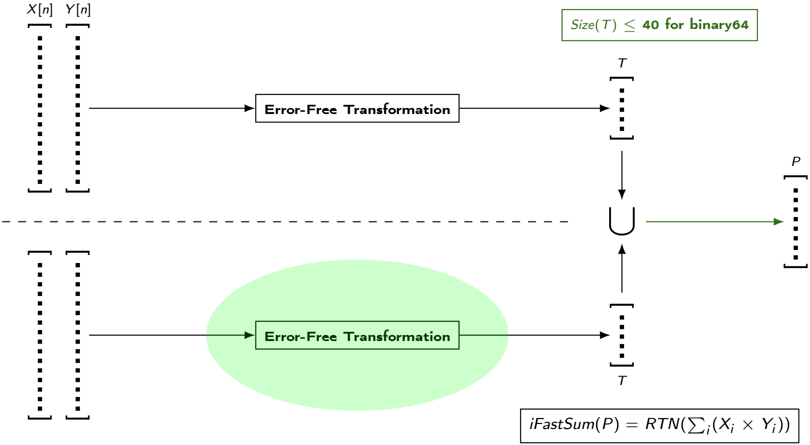
Parallel Dot Product



Parallel Dot Product



Parallel Dot Product



HybridSum (Zhu and Hayes, 2009) and OnlineExact (Zhu and Hayes, 2010)

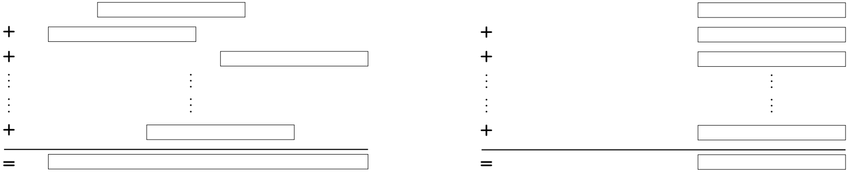
About Algorithms

- 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.

HybridSum (Zhu and Hayes, 2009) and OnlineExact (Zhu and Hayes, 2010)

About Algorithms

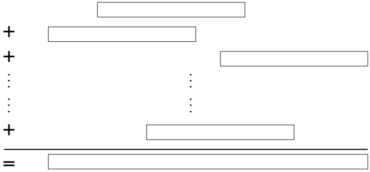
- Accumulate together the elements with the same exponent.
- Extra precision is simulated in two ways.
  - Split the input so the standard numbers are considered as accumulators.
  - Use quadruple precision.

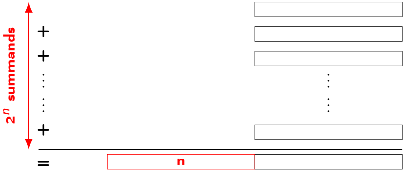


HybridSum (Zhu and Hayes, 2009) and OnlineExact (Zhu and Hayes, 2010)

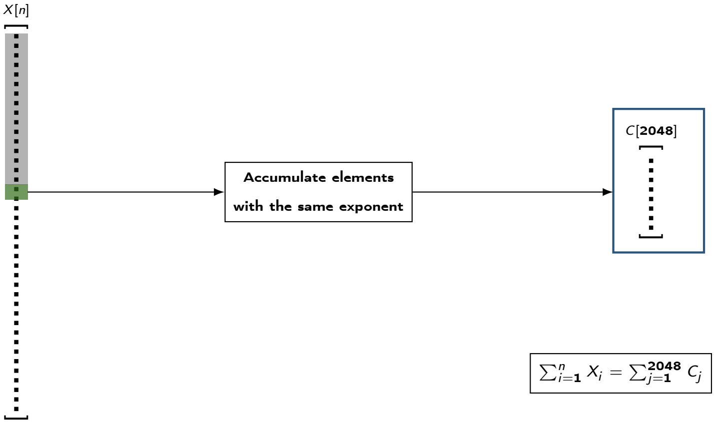
About Algorithms

- 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.



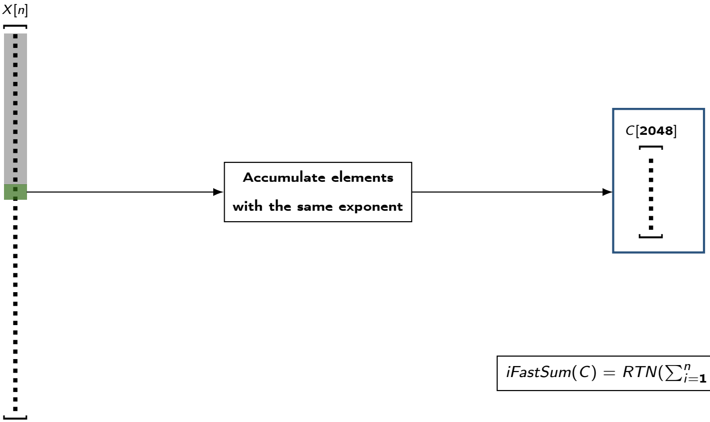


Algorithm OnlineExact (Zhu and Hayes, 2010)



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

Algorithm OnlineExact (Zhu and Hayes, 2010)

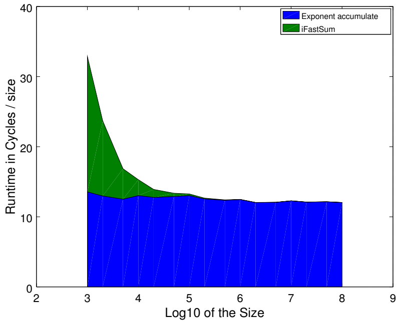


$$iFastSum(C) = RTN(\sum_{i=1}^{n} X_i)$$

Efficiency of the Algorithm OnlineExact

Implementation

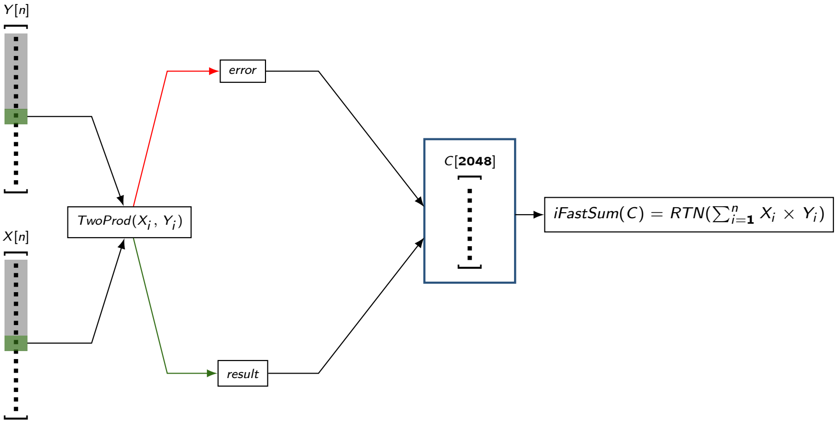
- Entry vector condition number $= 10^{16}$.



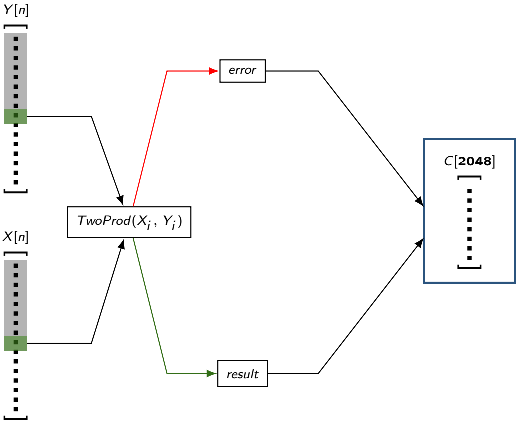
Note

- 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.

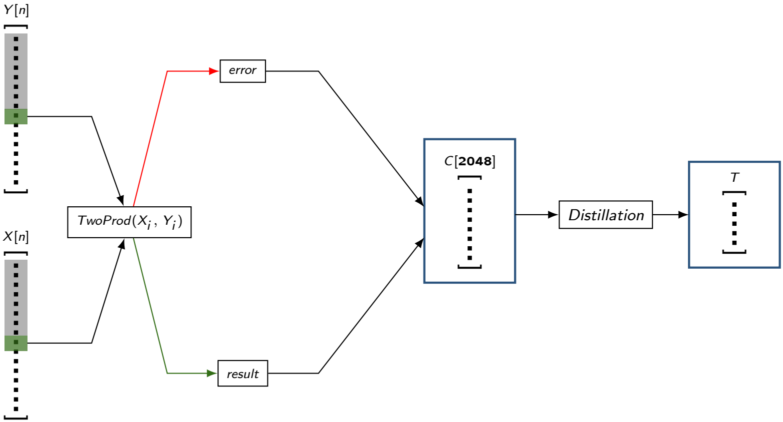
Error-Free Transformation



Error-Free Transformation



Error-Free Transformation



Euclidean Norm

How to Get a Reproducible Euclidean Norm

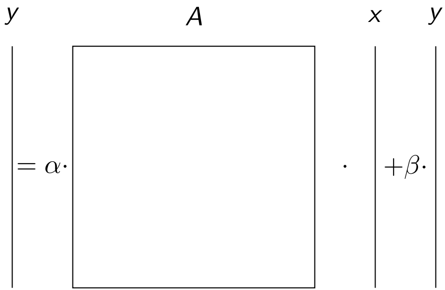
- Nrm2(X) $= \sqrt{\sum_{i=1}^{n} X_i^2}$.
- If $\sum_{i=1}^{n} X_i^2$ is correctly rounded, $\sqrt{\sum_{i=1}^{n} X_i^2}$ is faithfully rounded (Graillat and al., 2014).
- Dot(X,X) $= \sum_{i=1}^{n} X_i^2$ is correctly rounded.
- Nrm2(X) $= \sqrt{Dot(X, X)}$ is faithfully rounded and reproducible.

Sum of Absolute Values

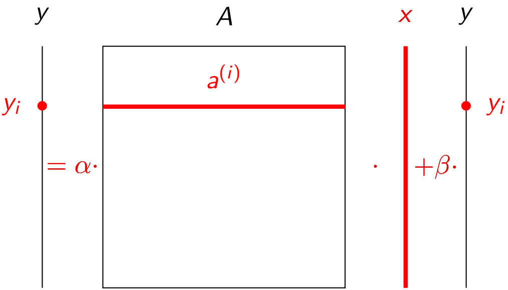
How to Get a Faithfully Rounded Sum of Absolute Values

- 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.

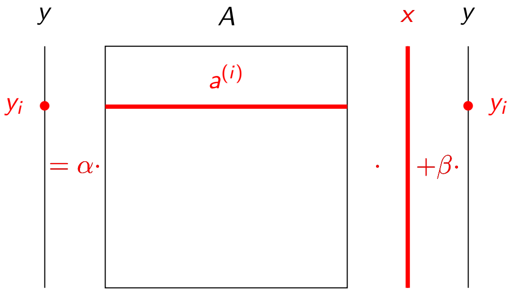
Matrix Vector Multiplication



Matrix Vector Multiplication



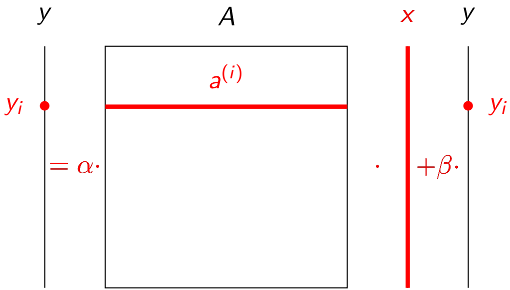
Matrix Vector Multiplication



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

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

Table of Contents

Experimental Framework

Shared Memory

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

Distributed Memory

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

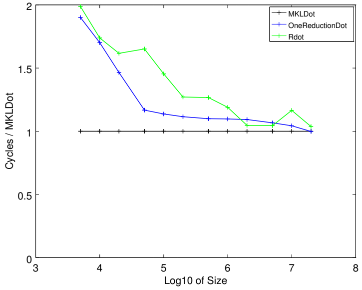
Accelerator

- Intel Xeon Phi 7120 accelerator, 60 cores.

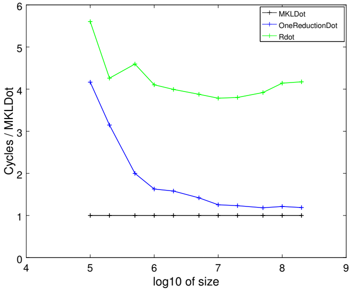
Results for Dot Product

Implementation

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



Shared Memory Performance



Xeon Phi Performance

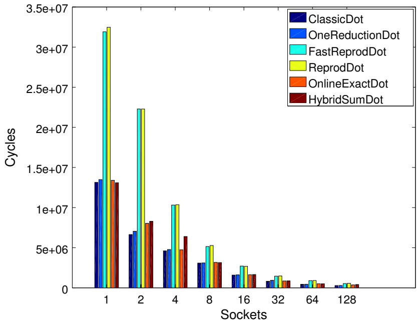
Results for Dot Product

Configurations

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

Dataset

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



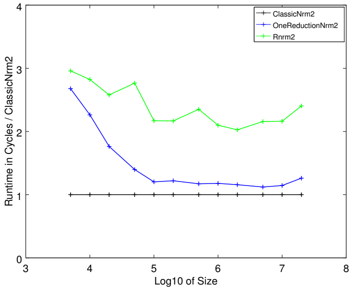
Note

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

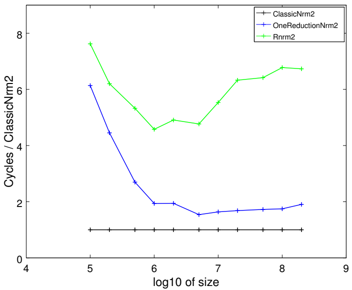
Results for Euclidean Norm

Implementation

- Manually optimized version for all algorithms.



Shared Memory Performance

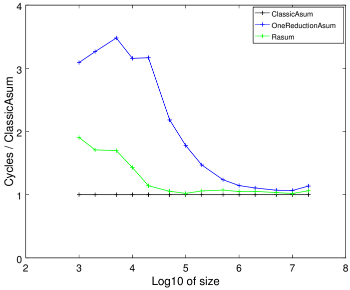


Xeon Phi Performance

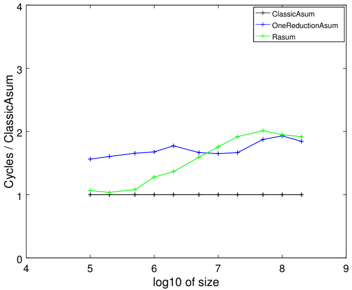
Results for Sum of Absolute Values

Implementation

- Manually optimized version for all algorithms.



Shared Memory Performance

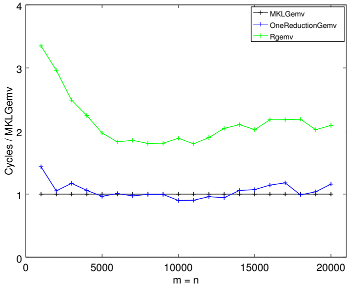


Xeon Phi Performance

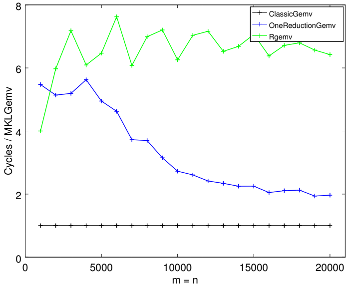
Results for Matrix Vector Multiplication

Implementation

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

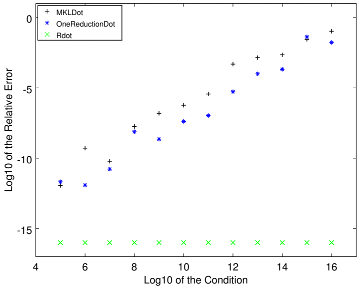


Shared Memory Performance

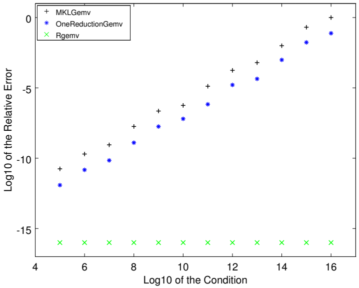


Xeon Phi Performance

Accuracy Results



Accuracy of Dot Product (size = 10^5)



Accuracy of Gemv (m = n = 1000)

Table of Contents

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.

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

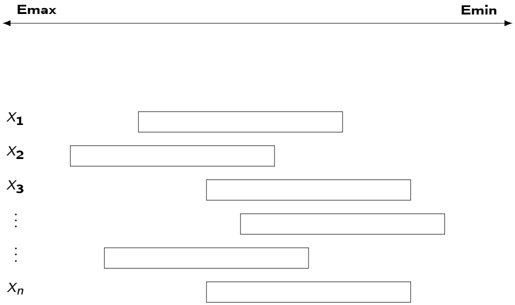
FOR

YOUR ATTENTION

ReprodSum and FastReprodSum (Demmel and Nguyen, 2013)

About Algorithms

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



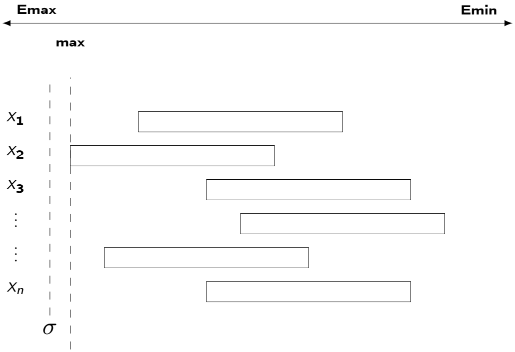
Steps for Sequential

- Compute max.

ReprodSum and FastReprodSum (Demmel and Nguyen, 2013)

About Algorithms

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



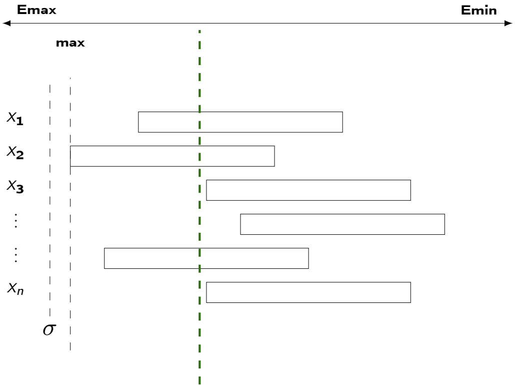
Steps for Sequential

- Compute max.

ReprodSum and FastReprodSum (Demmel and Nguyen, 2013)

About Algorithms

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



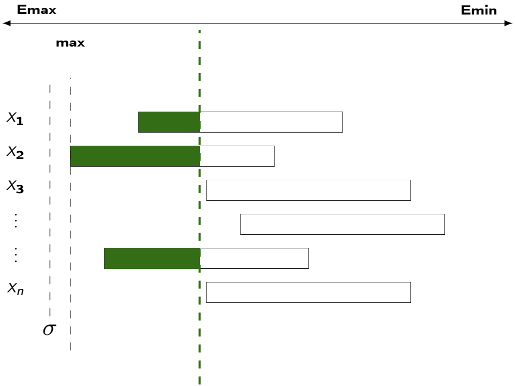
Steps for Sequential

- Compute max.

ReprodSum and FastReprodSum (Demmel and Nguyen, 2013)

About Algorithms

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



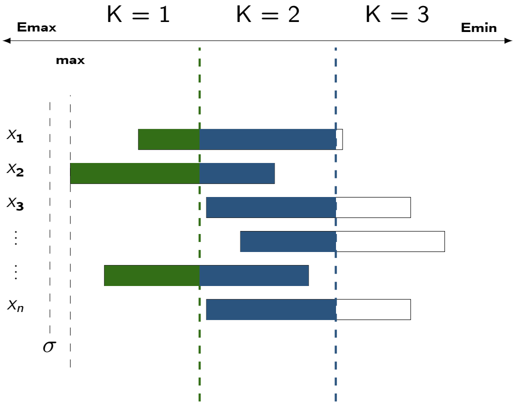
Steps for Sequential

- Compute max.
- Sum each fold.

ReprodSum and FastReprodSum (Demmel and Nguyen, 2013)

About Algorithms

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



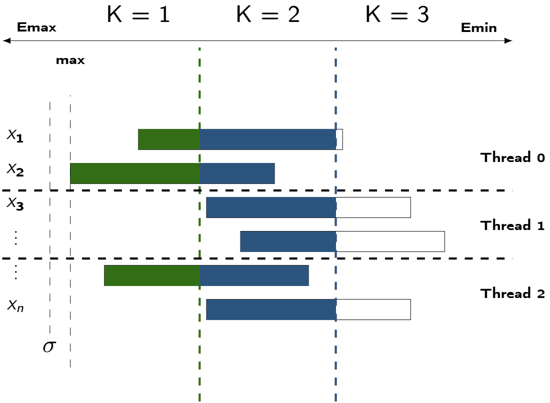
Steps for Sequential

- Compute max.
- Sum each fold.

ReprodSum and FastReprodSum (Demmel and Nguyen, 2013)

About Algorithms

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



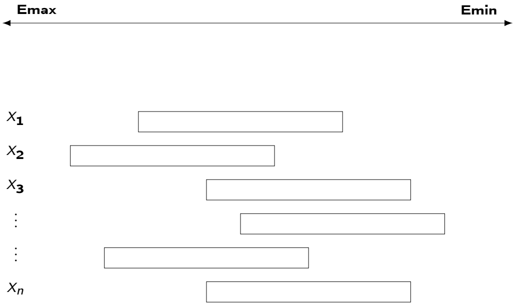
Steps for Sequential

- Compute max.
- Sum each fold.

Steps for Parallel

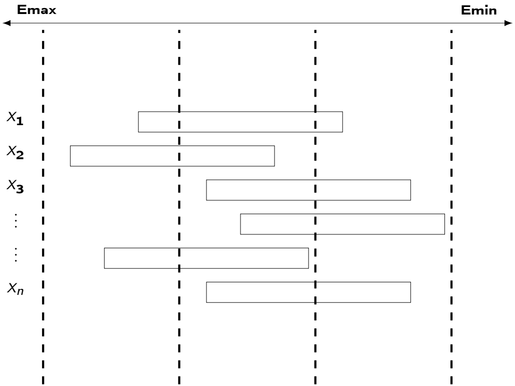
- Max (Compute).
- Max (Reduce).
- Sum (Compute).
- Sum (Reduce).

OneReduction (Demmel and Nguyen, 2013)



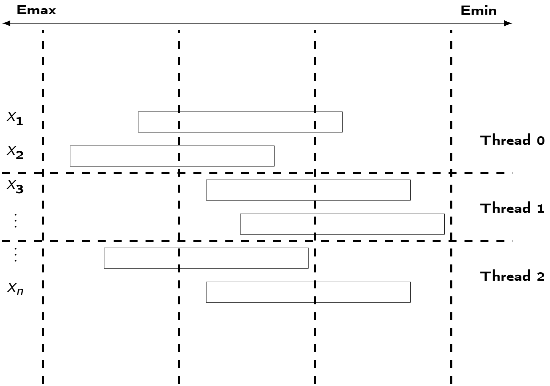
Steps for Parallel

OneReduction (Demmel and Nguyen, 2013)



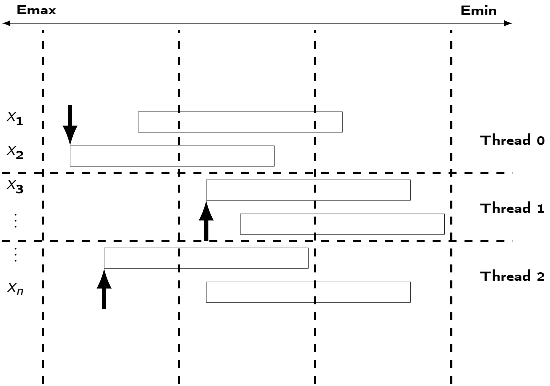
Steps for Parallel

OneReduction (Demmel and Nguyen, 2013)



Steps for Parallel

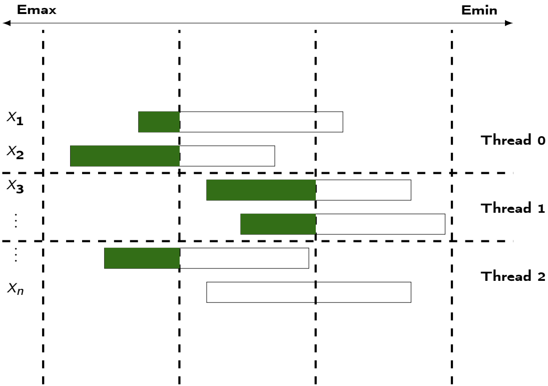
OneReduction (Demmel and Nguyen, 2013)



Steps for Parallel

- Max (Compute).

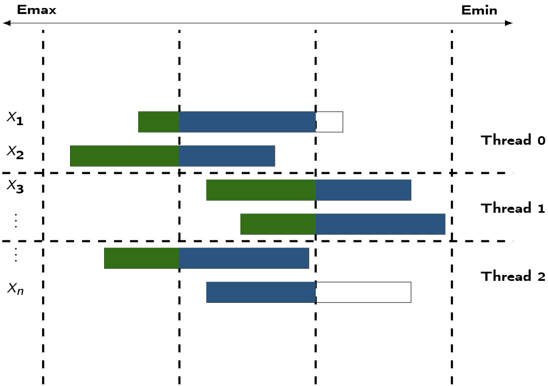
OneReduction (Demmel and Nguyen, 2013)



Steps for Parallel

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

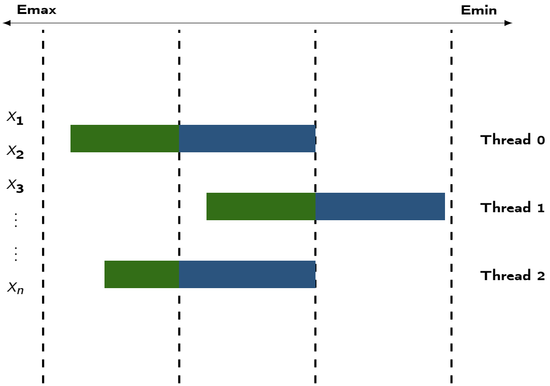
OneReduction (Demmel and Nguyen, 2013)



Steps for Parallel

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

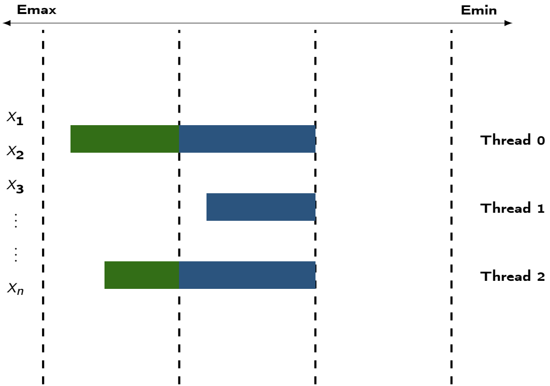
OneReduction (Demmel and Nguyen, 2013)



Steps for Parallel

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

OneReduction (Demmel and Nguyen, 2013)



Steps for Parallel

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

OneReduction (Demmel and Nguyen, 2013)

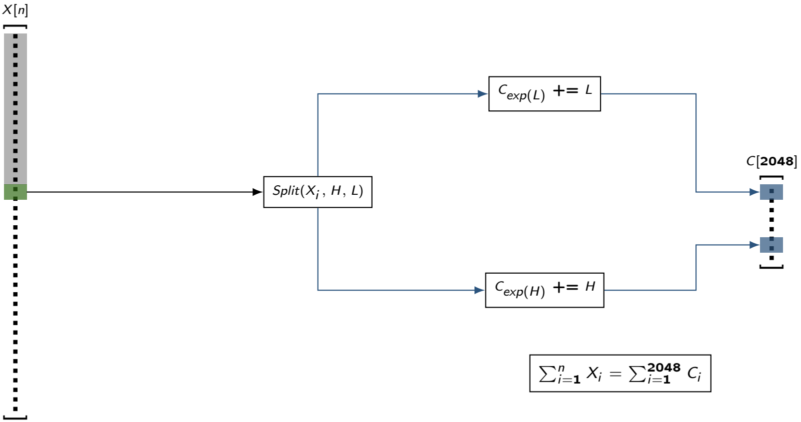
Pros

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

Cons

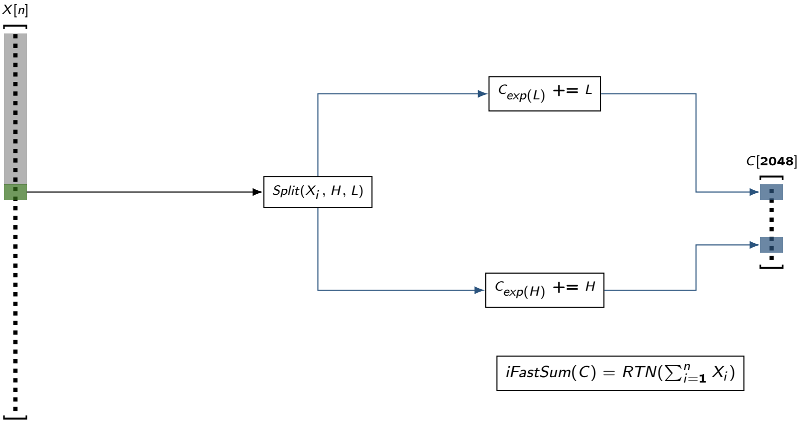
- Accuracy problem on ill-conditioned sums.

Algorithm HybridSum (Zhu and Hayes, 2009)

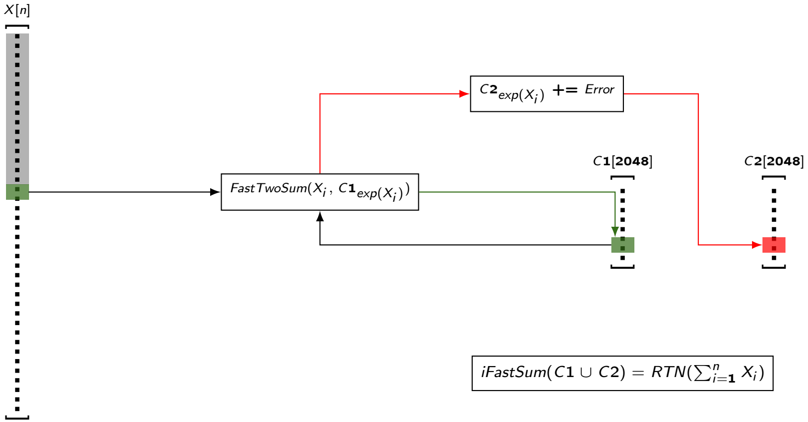


$$\sum_{i=1}^{n} X_i = \sum_{i=1}^{2048} C_i$$

Algorithm HybridSum (Zhu and Hayes, 2009)



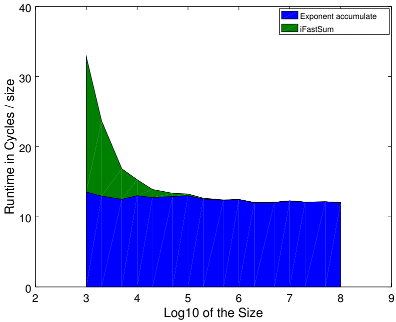
Algorithm OnlineExact (Zhu and Hayes, 2010)



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.