RARE-BLAS

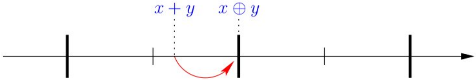
PRECIS 2017

Chemseddine Chohra, Philippe Langlois, Rafife Nheili, and David Parello

DALI, LIRMM, University of Perpignan, France

Numerical reproducibility issue

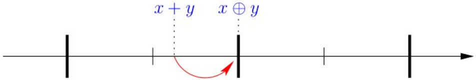
- Limited machine precision.
  - Using floating point number as



    approximation.
  - $x \longrightarrow X = round(x)$ if $x \notin \mathbb{F}$ or x if $x \in \mathbb{F}$.
  - $X + Y \neq X \oplus Y = round(X + Y)$.
- Non-associativity of addition.
  - $A \oplus (B \oplus C) \neq (A \oplus B) \oplus C$.
  - For instance : $M = 2^{53}$; $(-M \oplus M) \oplus 1 \neq -M \oplus (M \oplus 1)$.
- Reordering of floating-point operations:
  - Parallelization (vectorization, multi-threading, ...)
  - Dynamic scheduling

Numerical reproducibility issue

- Limited machine precision.
    - Using floating point number as



      approximation.
    - $x \longrightarrow X = round(x)$ if $x \notin \mathbb{F}$ or x if $x \in \mathbb{F}$.
    - $X + Y \neq X \oplus Y = round(X + Y)$.
- Non-associativity of addition.
    - $A \oplus (B \oplus C) \neq (A \oplus B) \oplus C$.
    - For instance : $M = 2^{53}$; $(-M \oplus M) \oplus 1 \neq -M \oplus (M \oplus 1)$.
- Reordering of floating-point operations:
    - Parallelization (vectorization, multi-threading, ...)
    - Dynamic scheduling
- Numerical Reproducibility: Getting bit-wise identical result for every p-parallel run

Overview

RARE-BLAS

Objectives:

- Guarantee numerical reproducibility
- maximum **Accuracy**
- maximum **Performance**

RARE-BLAS

Objectives:

- Guarantee numerical reproducibility
- maximum **Accuracy**
- maximum **Performance**

Methodology:

- Correctly Rounded solution $\rightarrow$ unique solution $\rightarrow$ reproducibility.

RARE-BLAS

Objectives:

- Guarantee numerical reproducibility
- maximum **Accuracy**
- maximum **Performance**

Methodology:

- Correctly Rounded solution $\rightarrow$ unique solution $\rightarrow$ reproducibility.

Reproducible level 1 and 2 BLAS functions: Rnrm2, Rasum, Rdot, Rgemv, Rtrsv

RARE-BLAS

Objectives:

- Guarantee numerical reproducibility
- maximum **Accuracy**
- maximum **Performance**

Methodology:

- Correctly Rounded solution $\rightarrow$ unique solution $\rightarrow$ reproducibility.

Reproducible level 1 and 2 BLAS functions: Rnrm2, Rasum, Rdot, Rgemv, Rtrsv

Implementations target Intel CPU micro-architectures.

Useful algorithms

Error-Free Transformations:

- Algorithm TwoProd(T. J. Dekker, 1971)
- Algorithm TwoSum (Knuth, 1998)
- Distillation (I. J. Anderson, 2006)

Accurate summations:

- Accurate: Sum-K[1]
- Faithful: AccSum[2], FastAccSum[3]
- Correctly rounded (in RtN): iFastSum, HydribSum[4], OnlineExact sum[5]

[1] Takeshi Ogita, Siegfried M. Rump, and Shin'ichi Oishi. "Accurate sum and dot product". In: 26.6 (2005), pp. 1955–1988.

[2] Siegfried M. Rump, Takeshi Ogita, and Shin'ichi Oishi. "Accurate floating-point summation – Part I: Faithful rounding". In: 31.1 (2008), pp. 189–224.

[3] Siegfried M. Rump. "Ultimately fast accurate summation". In: 31.5 (2009), pp. 3466–3502.

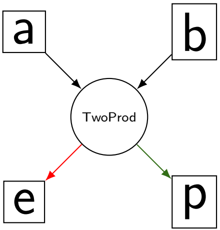
[4] Yong-Kang Zhu and Wayne B. Hayes. "Correct rounding and hybrid approach to exact floating-point summation". In: *SIAM J. Sci. Comput.* 31.4 (2009), pp. 2981–3001. ISSN: 1064-8275. DOI: 10.1137/070710020. URL: http://dx.doi.org/10.1137/070710020.

[5] Yong-Kang Zhu and Wayne B. Hayes. "Algorithm 908: Online Exact Summation of Floating-Point Streams". In: 37.3 (Sept. 2010), 37:1–37:13. URL: http://doi.acm.org/10.1145/1824801.1824815.

Error-Free Transformations

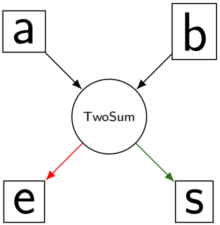
Algorithm TwoProd(T. J. Dekker, 1971)

- Input: a, b.
- Output: p, e.
  - $p = a \otimes b = round(a \times b)$.
  - $p + e = a \times b$.



Algorithm TwoSum (Knuth, 1998)

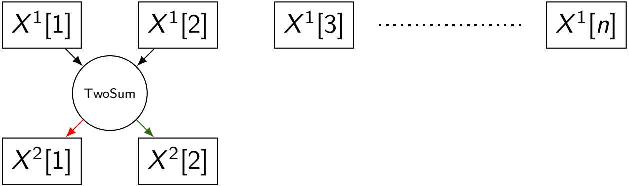
- Input: a, b.
- Output: s, e.
  - $s = a \oplus b = round(a + b)$.
  - $s + e = a + b$.



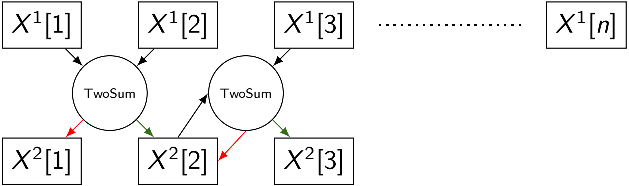
Distillation (I. J. Anderson, 2006)

$\boxed{X^1[1]}$ $\boxed{X^1[2]}$ $\boxed{X^1[3]}$ .... $\boxed{X^1[n]}$

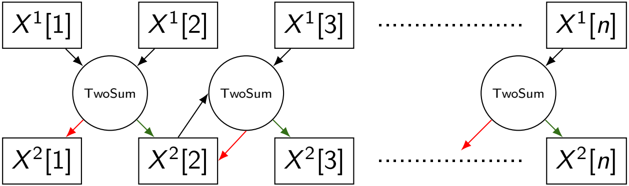
Distillation (I. J. Anderson, 2006)



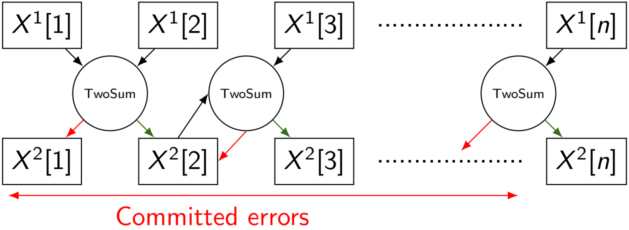
Distillation (I. J. Anderson, 2006)



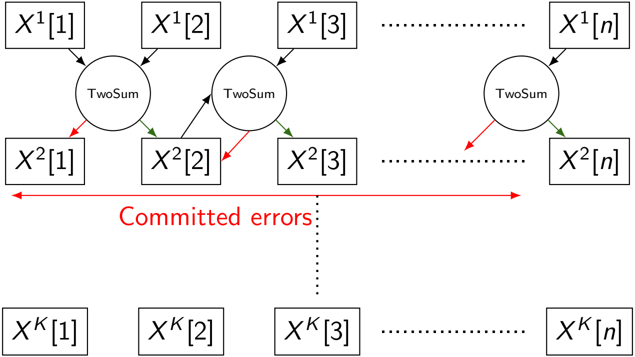
Distillation (I. J. Anderson, 2006)



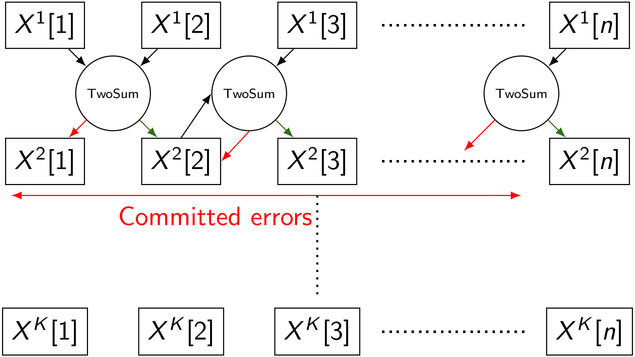
Distillation (I. J. Anderson, 2006)



Distillation (I. J. Anderson, 2006)

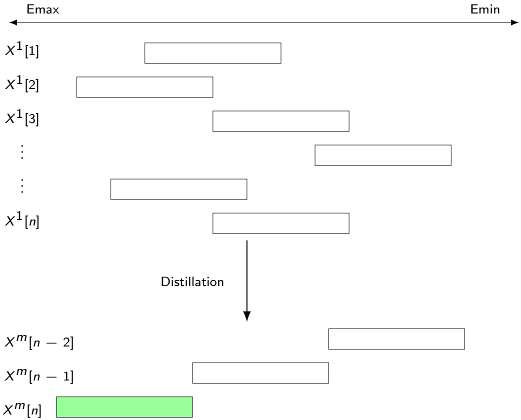


Distillation (I. J. Anderson, 2006)

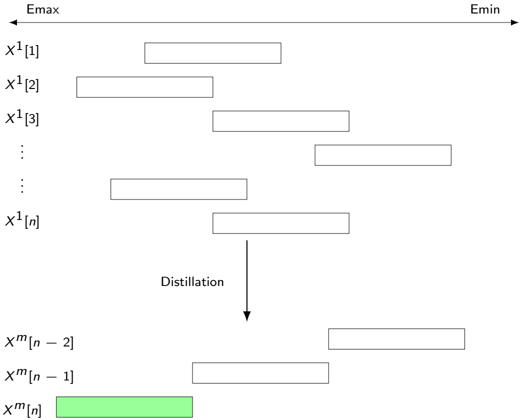


$\Rightarrow$ Sum-K

iFastSum (Zhu and Hayes, 2009)



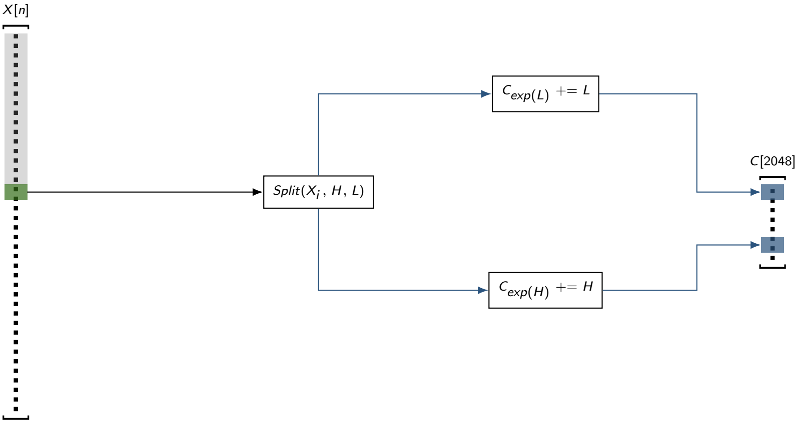
iFastSum (Zhu and Hayes, 2009)



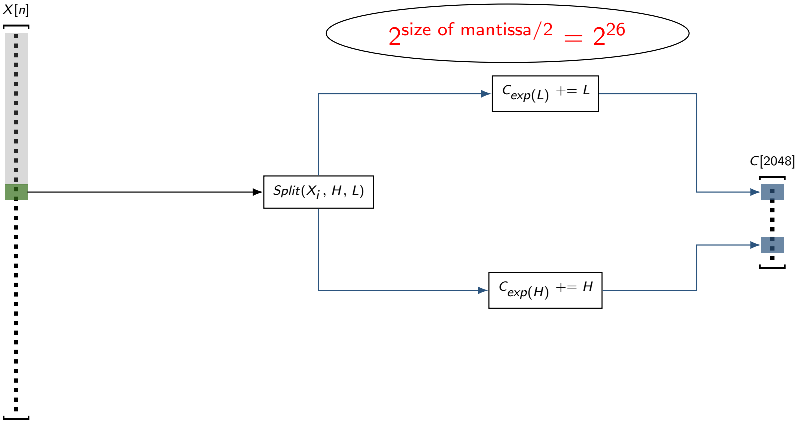
Main drawback

- Requires several passes through the input vector.

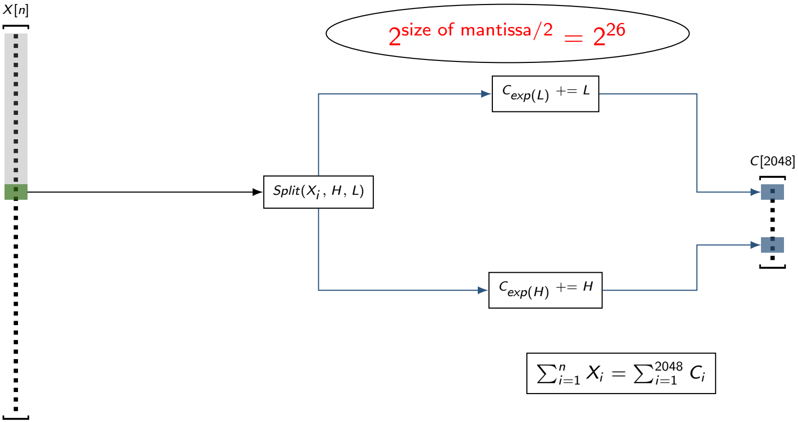
HybridSum (Zhu and Hayes, 2009)



HybridSum (Zhu and Hayes, 2009)



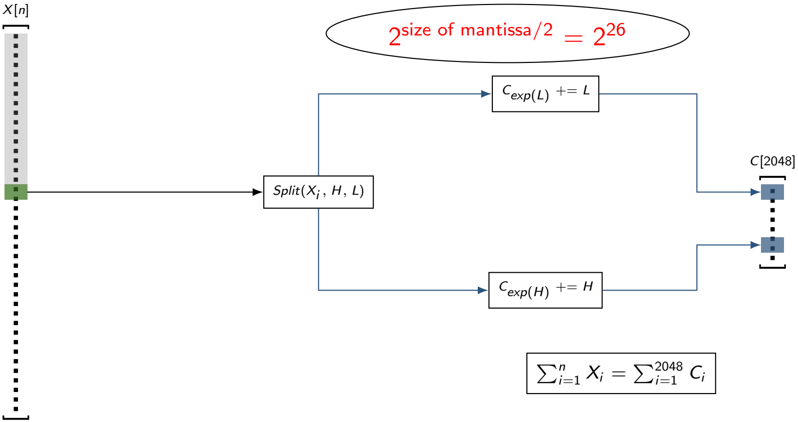
HybridSum (Zhu and Hayes, 2009)



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

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

HybridSum (Zhu and Hayes, 2009)



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

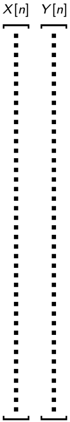
$$iFastSum(C) = RTN(\sum_{i=1}^{n} X_i)$$

Main drawback

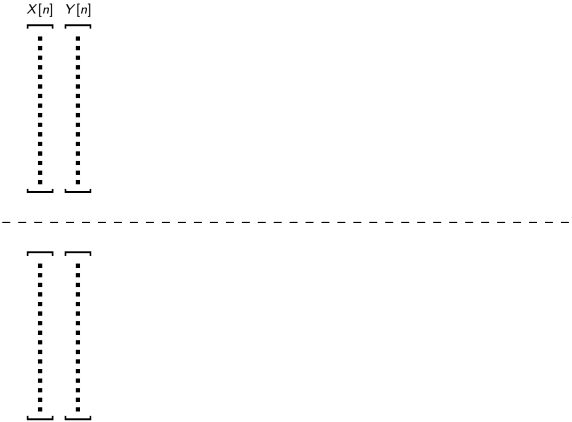
- No vectorization for accumulation.

Overview

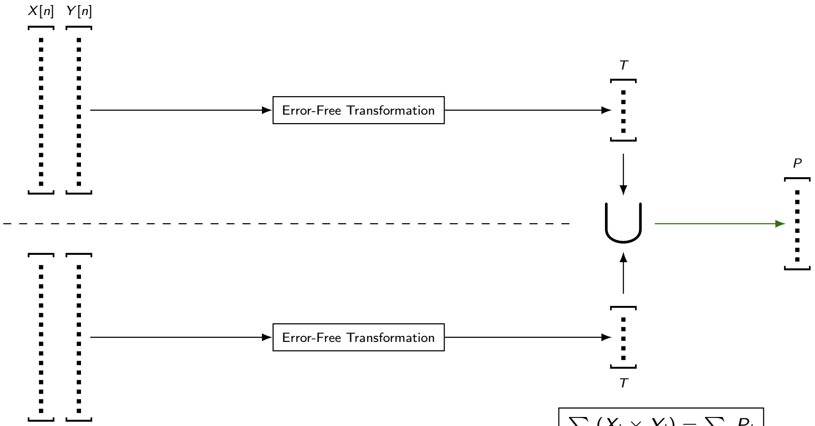
Parallel RDot



Parallel RDot



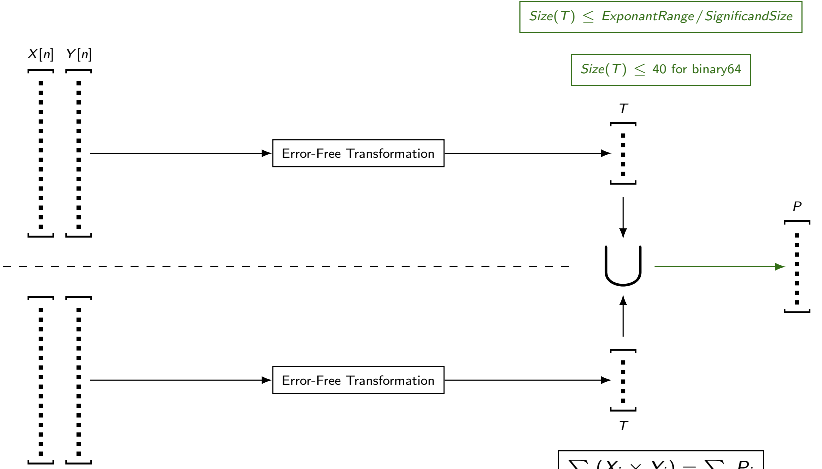
Parallel RDot



$$\boxed{\sum_i (X_i \times Y_i) = \sum_j P_j}$$

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

Parallel RDot



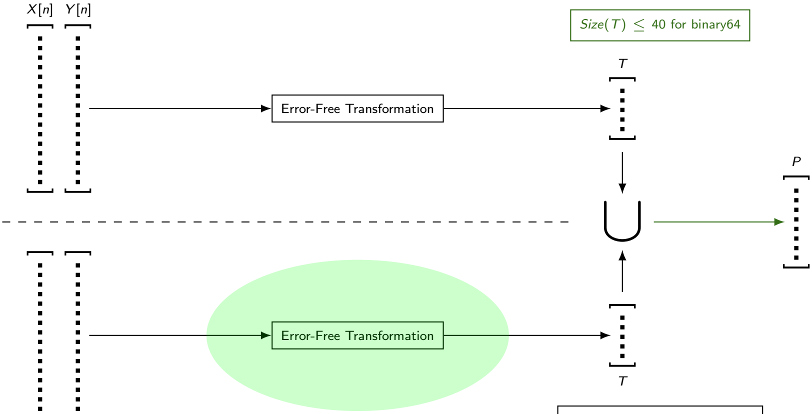
$$\boxed{\textstyle\sum_i (X_i \times Y_i) = \sum_j P_j}$$

$$\boxed{iFastSum(P) = RTN(\textstyle\sum_i (X_i \times Y_i))}$$

Parallel RDot

$Size(T) \leq ExponantRange\,/\,SignificandSize$

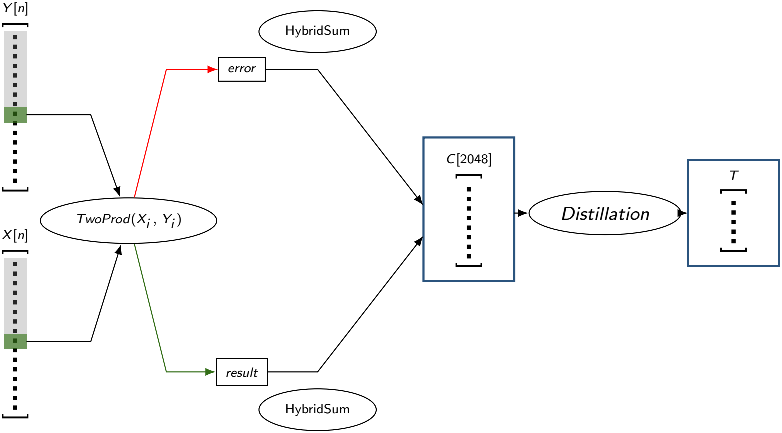
$Size(T) \leq 40$ for binary64



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

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

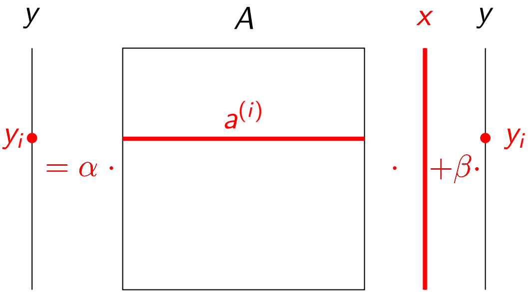
Parallel RDot: Error-Free Transformation



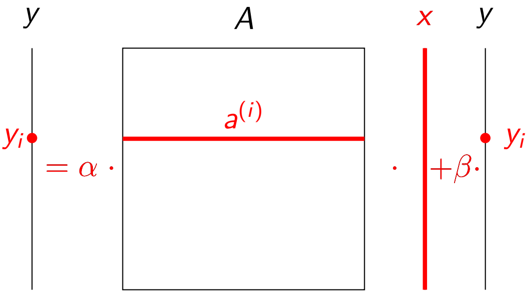
Parallel Rgemv

$$y = \alpha \cdot A \cdot x + \beta \cdot y$$

Parallel Rgemv



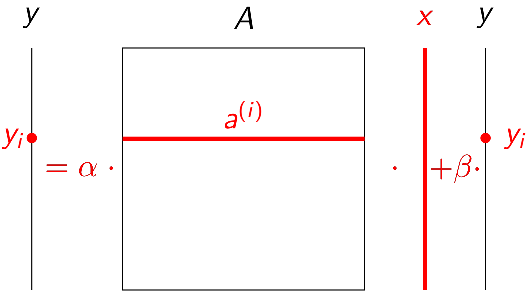
Parallel Rgemv



Algorithm Steps

- $y_i = \alpha \cdot (a^{(i)} \cdot x) + \beta \cdot y_i$
- $T1 = EFT(a^{(i)} \cdot x)$
- $T2 = EFT(T1 \cdot \alpha)$
- $(T2_K, T2_{K+1}) = TwoProd(\beta \cdot y_i)$
- $y_i = iFastSum(T2) \Rightarrow y_i = RTN(\alpha \cdot (a^{(i)} \cdot x) + \beta \cdot y_i)$

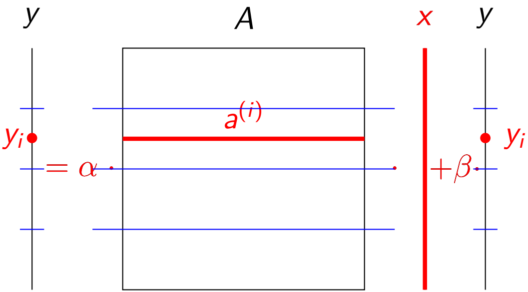
Parallel Rgemv



Algorithm Steps

- $y_i = \alpha \cdot (a^{(i)} \cdot x) + \beta \cdot y_i$
- $T1 = EFT(a^{(i)} \cdot x)$
- $T2 = EFT(T1 \cdot \alpha)$
- $(T2_K, T2_{K+1}) = TwoProd(\beta \cdot y_i)$
- $y_i = iFastSum(T2) \Rightarrow y_i = RTN(\alpha \cdot (a^{(i)} \cdot x) + \beta \cdot y_i)$

Parallel Rgemv



Algorithm Steps

- $y_i = \alpha \cdot (a^{(i)} \cdot x) + \beta \cdot y_i$
- $T1 = EFT(a^{(i)} \cdot x)$
- $T2 = EFT(T1 \cdot \alpha)$
- $(T2_K, T2_{K+1}) = TwoProd(\beta \cdot y_i)$
- $y_i = iFastSum(T2) \Rightarrow y_i = RTN(\alpha \cdot (a^{(i)} \cdot x) + \beta \cdot y_i)$

Overview

Experimental Framework: machines

Shared Memory

- dual Xeon E5-2650 v2 16 cores (8 per socket).
- L1/L2 = 32/256 KB.
- Bandwidth = 59.7 GB/s.

Accelerator

- Intel Xeon Phi 7120 accelerator, 60 cores.
- L1/L2 = 32/512 KB.
- Bandwidth = 352 GB/s.

Distributed Memory

- OCCIGEN (64^{th} supercomputer in top500 list).
- 4212 Xeon E5-2690 v3 socket (12 cores per socket).
- L1/L2 = 32/256 KB.
- Bandwidth = 68 GB/s.

Experimental Framework: optimizations

Compiler optimizations

- `-O3 -fp-model double -fp-model strict -funroll-all-loops`

  `-fp-model double : Rounds to 53-bit precision.`
  `-fp-model strict : Disable contractions.`

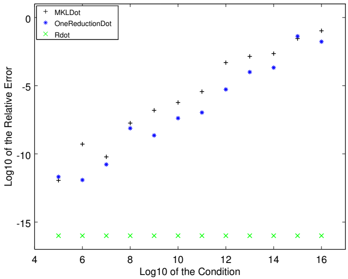
Manual optimizations

- Vectorization (Intel intrinsics)
- Loop unrolling
- Data prefetching
- + *Algorithmic optimizations*

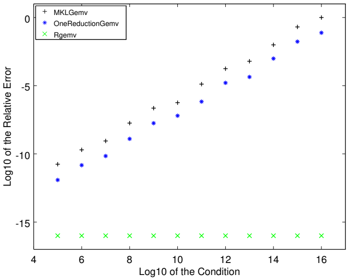
Baseline and other algorithm

- MKLDot, MKLGemv (Intel MKL)
- OneReductionDot, OneReductionGemv (based on OneReduction[a])

[a] Demmel and Nguyen, 2013



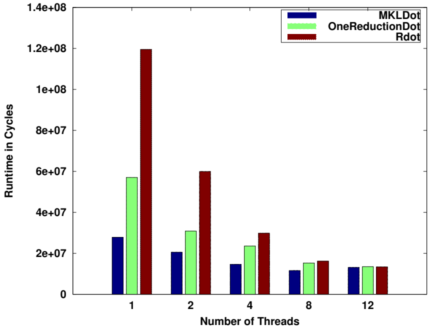
Accuracy of Dot (size $= 10^5$)



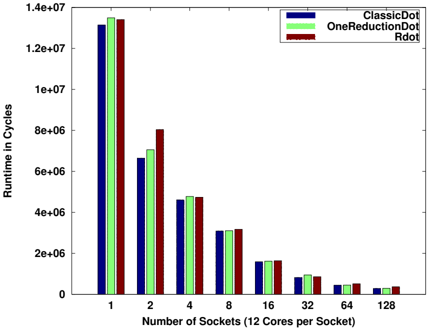
Accuracy of Gemv ($m = n = 1000$)

Dot: Distributed memory

Entry size $= 10^7$ / Entry condition number $= 10^{32}$.



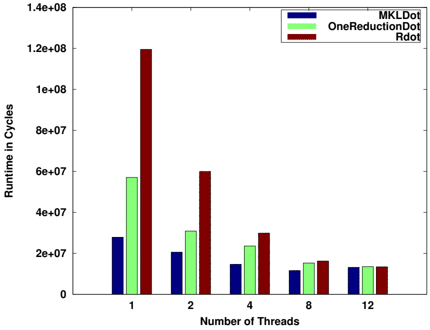
Distributed memory: single socket



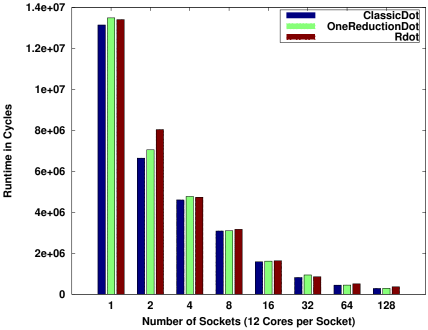
Distributed memory: multi-sockets

Dot: Distributed memory

Entry size $= 10^7$ / Entry condition number $= 10^{32}$.



Distributed memory: single socket

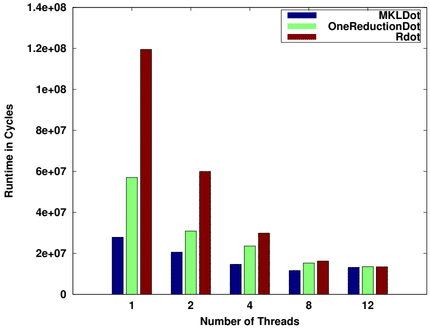


Distributed memory: multi-sockets

Memory latency hides extra computations

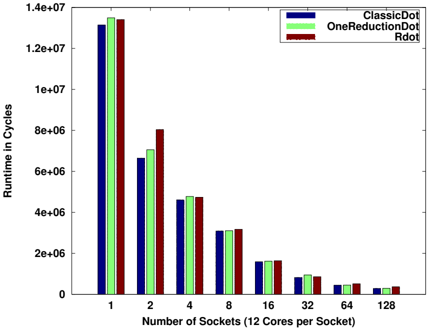
Dot: Distributed memory

Entry size $= 10^7$ / Entry condition number $= 10^{32}$.



Distributed memory: single socket

Memory latency hides extra computations

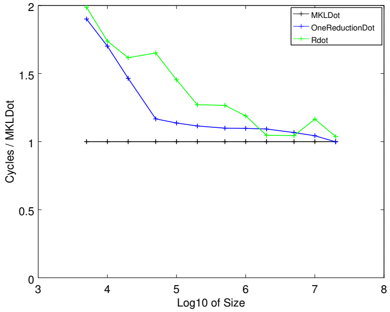


Distributed memory: multi-sockets

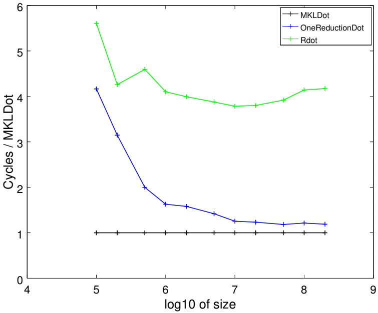
EFT unions ⇒ limited impact on communications

Dot: Shared memory and Accelerator

Entry condition number $= 10^8$.



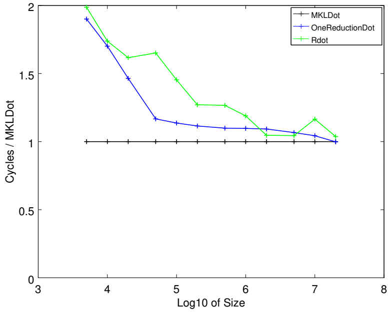
Shared Memory (16 cores)



Accelerator (Xeon Phi)

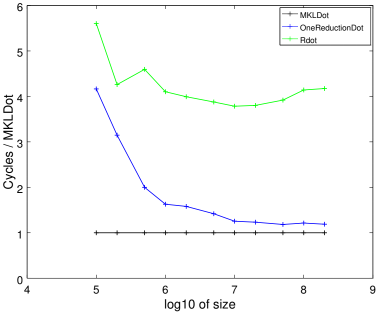
Dot: Shared memory and Accelerator

Entry condition number $= 10^8$.



Shared Memory (16 cores)

Sandy bridge: $2 \times$ VPU.
AVX (256bits)

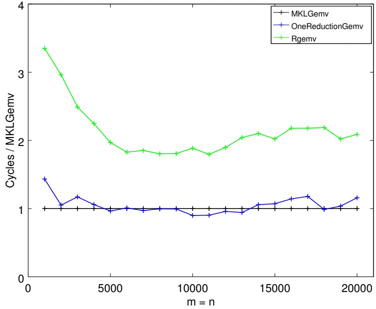


Accelerator (Xeon Phi)

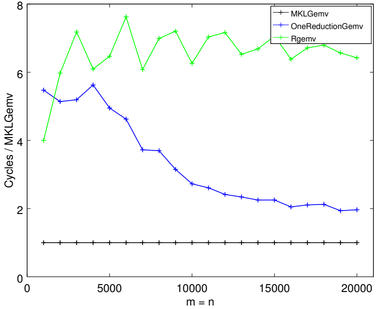
Knights corner: $1 \times$ VPU.
AVX512 (512bits)

Gemv: Shared memory and Accelerator

Entry condition number $= 10^8$.



Shared Memory (16 cores)



Accelerator (Xeon Phi)

Impact of optimizations (HybridSum)

```
ALGORITHM HybridSum (naive).
INPUT : A, an array of floating point
summands.
OUTPUT : S, the correctly rounded sum of A.
BEGIN.
  1 Declare an intermediate array C.
  2 FOREACH element of A as a do.
      1 split(a, a_h, a_l).
      2 i = exponent(a_h).
      3 C_i += a_h.
      4 i = exponent(a_l).
      5 C_i += a_l.
    END FOREACH.
  3 RETURN iFastSum(C).
END.
```

Impact of optimizations (HybridSum)

```
ALGORITHM HybridSum (naive).
INPUT : A, an array of floating point
summands.
OUTPUT : S, the correctly rounded sum of A.
BEGIN.
  1 Declare an intermediate array C.
  2 FOREACH 8 elts of A as a do (step1)
      1 split(a, a_h, a_l).
      2 i = exponent(a_h).
      3 C_i += a_h.
      4 i = exponent(a_l).
      5 C_i += a_l.
    END FOREACH.
  3 RETURN iFastSum(C).
END.
```

Impact of optimizations (HybridSum)

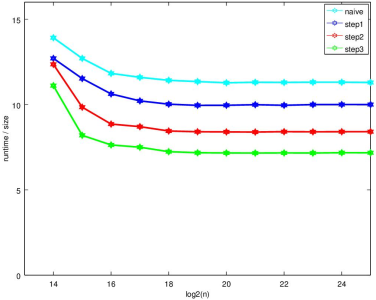
```
ALGORITHM HybridSum (naive).
INPUT : A, an array of floating point
summands.
OUTPUT : S, the correctly rounded sum of A.
BEGIN.
  1 Declare an intermediate array C.
  2 FOREACH 8 elts of A as a do (step1)
      1 prefetch data (step2).
      2 split(a, a_h, a_l).
      3 i = exponent(a_h).
      4 C_i += a_h.
      5 i = exponent(a_l).
      6 C_i += a_l.
    END FOREACH.
  3 RETURN iFastSum(C).
END.
```

Impact of optimizations (HybridSum)

```
ALGORITHM HybridSum (naive).
INPUT : A, an array of floating point
summands.
OUTPUT : S, the correctly rounded sum of A.
BEGIN.
  1 Declare an intermediate array C.
  2 FOREACH 8 elts of A as a do (step1)
      1 prefetch data (step2).
      2 split(a, a_h, a_l).
      3 i = exponent(a_h).
      4 C_i += a_h.
      5 i = i - 27.  (step3)
      6 C_i += a_l.
    END FOREACH.
  3 RETURN iFastSum(C).
END.
```

Impact of optimizations (HybridSum)

```
ALGORITHM HybridSum (naive).
INPUT : A, an array of floating point
summands.
OUTPUT : S, the correctly rounded sum of A.
BEGIN.
  1 Declare an intermediate array C.
  2 FOREACH 8 elts of A as a do (step1)
      1 prefetch data (step2).
      2 split(a, a_h, a_l).
      3 i = exponent(a_h).
      4 C_i += a_h.
      5 i = i - 27.   (step3)
      6 C_i += a_l.
    END FOREACH.
  3 RETURN iFastSum(C).
END.
```



$\approx$ 40% saved

Conclusions

RARE-BLAS

- Reproducible level 1 and 2 BLAS. New functions: Rnrm2, Rasum, Rdot, Rgemv, Rtrsv
- High accuracy.
- Acceptable performances.

Conclusions

RARE-BLAS

- Reproducible level 1 and 2 BLAS. New functions: Rnrm2, Rasum, Rdot, Rgemv, Rtrsv
- High accuracy.
- Acceptable performances.

Future works

- Clean the code, write the doc and put it online!
- Extend to level 3 BLAS (different approach?)
- Improve the process (auto-tuning, meta-programming, ...)

Thank you. Questions?