Reproducible BLAS

Peter Ahrens, Jim Demmel, Hong Diep Nguyen,
Computer Science Division
UC Berkeley
bebop.cs.berkeley.edu/reproblas/
Outline

• Motivation for Reproducibility
• Design Goals (high level)
• Algorithm Sketch, for Sum
• Performance Data, for Sum
• Design Goals (details)
• Software structure and status
• More Performance Data, for ReproBLAS
• Testing
• Relation to IEEE 754 Standard
• Future ReproBLAS Interface – points to discuss
Motivation (1/3)

• Since roundoff makes floating point addition nonassociative, different orders of summation often give different answers
• On a parallel machine, the order of summation can vary from run to run, or even subroutine-call to subroutine-call, depending on scheduling of available resources, so answers can change
• Why is reproducibility important?
Motivation (2/3)

• Email on NA-Digest: Commercial finite-element SW vendor wanted a parallel reproducible sparse linear equation solver, because his customers (civil engineers) had contractual obligations to their customers to get the same answer from run to run: “Will the bridge fall down or not?”

• Responses from ~100 UC Berkeley faculty to email query about the importance of reproducibility:
  – Most common: How will I debug without reproducibility?
  – Rare: I know better, I do error analysis
  – How do I do fracture mechanics, where I do many random simulations looking for a very rare event, and when one occurs, I need to resimulate it exactly, while computing some side information?
  – What if my “illegal underground nuclear test detector” (funded by the United Nations) says “They did it!” and then “They didn’t do it”?
Motivation (3/3)

• Many workshops etc at recent SC meetings
  – gcl.cis.udel.edu/sc15bof.php

• Intel released MKL with CNR = Conditional Numerical Reproducibility
  – Guarantees determinism, if #cores fixed
  – Not scalable

• Used in climate modeling (CCSM, Pat Worley)

• Under discussion by 2 other standards committees:
  – IEEE 754 and Java
Design Goals for Reproducible Sum (high level)

1. Reproducible sum, independent of order, assuming a subset of IEEE 754
2. Accuracy at least as good as conventional, and tunable
3. Handle exceptions reproducibly
4. One read-only pass over summands
5. One reduction
6. Use as little memory as possible, to enable tiling BLAS
7. Modular design, for various use cases
Algorithm Sketch for Reproducible Summation

• Pre-Rounding: Simplest algorithm for reproducible sum \( s = \Sigma_i x(i) \)
  1. Compute \( M = \max_i |x(i)| \); exact and so reproducible
  2. Round all \( x(i) \) to 1 ulp (unit in last place) of \( M \); error introduced no worse than usual error bound
  3. Add rounded \( x(i) \); they behave like fixed point numbers so summation exact and so reproducible
    • Need \( \log_2 n \) extra overflow bits
• Drawback: costs 2 or 3 passes over data in serial, or 3 reduction/broadcast steps in parallel
  – Misses Goals 4 & 5
• Better: can do it in 1 pass, or 1 reduction, by interleaving all 3 steps
Pre-Rounding

emax

emin

x[1]

x[2]

x[3]

x[4]

x[5]

x[6]

...

8
Pre-Rounding

$max(|x|)$
Pre-Rounding

emax

emin

max(|x|)  Boundary

x[1]
x[2]
x[3]
x[4]
x[5]
x[6]
...

Bits discarded in advance
Pre-Rounding

- Costs 2 or 3 reduction/broadcast steps
Indexed Summation


- proc 1
- proc 2
- proc 3
Indexed Summation

• Boundaries predetermined

K = 2 bins
Indexed Summation

- Only keep top K bins, don’t compute or discard rest
Indexed Type

- $W = 40$ for doubles and $W = 13$ for floats
- Sums up to $2^{64}$ doubles or $2^{33}$ floats
- Carry once every $2^{11}$ doubles or $2^9$ floats
- Works for quad, half precision too
### Indexed Type

<table>
<thead>
<tr>
<th>Carry</th>
<th>Primary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

- Handles exceptions reproducibly
- Avoids unnecessary overflow
  - Special scaling on largest accumulators
- In ReproBLAS, K defaults to 3
Indexed Type → Floating Point

- New error bound for summing floating point numbers in order of strictly decreasing exponents
- Guarantees sum to 7 ulps
Performance results on 1024 proc Cray XC30
1.2x to 3.2x slowdown vs fastest (nonreproducible) code
dasum data for n=1M summands on up to p=1024 processors
3 reproducible sum algorithms compared, best one depends on n, p
code and papers at bebop.cs.berkeley.edu/reproblas
DG 1/7: Reproducible sum, independent of order, using a subset of IEEE 754

• Currently use Round-to-nearest-even (RNE), gradual underflow
  – Can extend to flush-to-zero (by pre-flushing)
• Inner loop similar to two-sum
  – \( s = Y_{pk} + (r|1) \), \( q = s - Y_{pk} \), \( Y_{pk} = s \), \( r = r - q \)
  – \( (r|1) \) makes rounding direction independent of \( Y_{pk} \), for reproducibility
  – In discussions with IEEE 754 to add two-sum with Round-to-nearest-ties-away-from-zero (RNA)
    • Could make ReproBLAS and XBLAS faster
• Use of one float for carries limits #summands to \( 2^{64} \) in double and \( 2^{33} \) in single (\( 2^{2p-W-2} \) in general)
DG 2/7: Accuracy at least as good as conventional sum, tunable

• Accuracy depends on
  – Width of one bin = W
  – Number of bins = K
  – #bits in “indexed sum” at least (K-1)*W
  – How “indexed sum” reduced to a single float

<table>
<thead>
<tr>
<th>Data type</th>
<th>Double</th>
<th>Single</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>40</td>
<td>13</td>
</tr>
<tr>
<td>K</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>(K-1)W</td>
<td>80</td>
<td>26</td>
</tr>
</tbody>
</table>
Error Bounds in Double (W=40,K=3)

- Notation:
  \[ T = \sum_i x(i), \quad S = \text{computed sum}, \quad M = \max_i |x(i)|, \quad \epsilon = 2^{-53} \]

- New error bound:
  \[ |S - T| \leq n \cdot 2^{-80} M + 7 \epsilon |T| \]

- Standard error bound:
  \[ |S - T| \leq n \epsilon \sum_i |x(i)| \leq n^2 \epsilon M \]

- New bound up to \(10^8\) x smaller when lots of cancellation (\(|T| \ll M\))
Relative Error Comparison

Condition number = \( n \cdot \max_i |x(i)| / \left| \sum_i x(i) \right| \)
DG 3/7: Handle Exceptions Reproducibly

• Challenge: add \(\{X, X, -X, -X, 1\}\) where \(X+X = +\text{Inf}\)
  – Depending on order, could get \(+\text{Inf}, -\text{Inf}, \text{NaN}, 0\) or \(1\)
  – Dot products harder: \([X,X,1,-X,-X] \ast [X,X,1,X,X]^T\)

• Case 1: Summands finite
  – Scale to avoid all intermediate overflows
  – Return +\text{Inf} or –\text{Inf} only if true result overflows

• Case 2: Some summands infinite or NaN
  – Return +\text{Inf}, -\text{Inf} or NaN following IEEE 754 rules

• Case 3: nrm2 (requires scaling)
DG 4/7: One read-only pass over data

• Natural requirement for good performance
  – Limit communication, memory required
  – Ex: Pre-Rounding approach to \( \text{dot}(x,y) \) would require storing or recomputing all \( x(i) \times y(i) \)

• Not all related work does it this way

DG 5/7: One reduction operation

• Ditto
DG 6/7: Use as little memory as possible, to enable tiling BLAS (1/2)

• Classical result for tiling $n \times n$ matmul $C = A \times B$ on machine with cache of $W$ words:
  
  – Thm: #words moved between main memory and cache $= \Omega(n^3 / W^{1/2})$
  
  – Bound attained by tiling $A, B, C$ into $W^{1/2} \times W^{1/2}$ tiles
  
  – Analogous result for parallel matmul

• What if each entry of $C$ represented by “reproducible accumulator” of $R$ words?
  
  – Thm: #words moved between main memory and cache $= \Omega(n^3 R^{1/2} / W^{1/2})$
  
  – Attained by tiles of size $(W/R)^{1/2} \times (WR)^{1/2} \times (W/R)^{1/2}$

• Extends to algorithms accessing arrays (eg n-body, ...)

25
DG 6/7: Use as little memory as possible, to enable tiling BLAS (2/2)

- What if each entry of C represented by “reproducible accumulator” of R words?
  - Thm: #words moved between main memory and cache = \( \Omega(n^3 R^{1/2} / W^{1/2}) \)
  - Attained by tiles of size \((W/R)^{1/2} \times (WR)^{1/2} \times (W/R)^{1/2}\)

- For our algorithm, \(R = 2K = 2 \times \#\text{bins}\)
- \(K=3\) so \(R=6\) in baseline double precision
- Related work uses much larger \(R\)
  - \(R = 67 \approx 2(\text{emax-emin})/64\) for Kulisch’s “exact dot product”
  - \(R = n\) in “Ultimately Fast Accurate Summation”, S. Rump
DG 7/7: Modular design, for various use cases

• “Complete reproducibility”:
  – Bitwise identical sum on any computer, any hardware resources, any scheduling, any input order
  – Ex: $fl(A*B) = fl((A*P)*(P^T*B))$
  – Out-of-scope:
    • saxpy, srot, srotm, ...: depends on FMA, div & sqrt
    • isamax: depends on order of search

• “Selective reproducibility”:
  – Only modify nonreproducible part of code
  – Ex: Dot product reproducible on one parallel platform, assuming same data layout, where only MPI reduction order may vary => only modify MPI reduction
Software Structure

• Details at bebop.cs.berkeley.edu/reproblas
• Primitive Operations
• Properties / Restrictions / Tradeoffs
  – Choosing $K = \#\text{bins}$, $W = \text{bin width}$
• Current status of ReproBLAS
Primitive Operations

• Bin i represented by \([Y_C(i), Y_P(i)]\), two floating point numbers
• \(I\) = Indexed type = tuple of K consecutive bins
• \(\text{Index}(I)\) = index of top bin in \(I\)
• \(\text{Index}(x)\) = index of top bin containing (part of) \(x\)
• \(\text{Update}(x, I)\) shifts top bin of \(I\) if needed to hold \(x\)
• \(\text{Deposit}(x, I)\) adds \(x\) to \(I\) assuming \(I\) can hold \(x\) (3K-2 flops)
• \(\text{Renorm}(I)\) renormalizes each \([Y_C(i), Y_P(i)]\) in \(I\)
• \(\text{AddFloatToIndexed}(x, I)\) does:
  – \(\text{Update}(x, I)\), \(\text{Deposit}(x, I)\), \(\text{Renorm}(I)\)
• \(\text{Sum}([x_1, \ldots, x_n], I)\)
  – Only does Update, Renorm every \(2^9\) or \(2^{11}\) Deposits
• \(\text{AddIndexedToIndexed}(I, J)\) … for parallel reduction
• \(\text{ConvertIndexedToFloat}(I, x)\)
## Properties/Restrictions/Tradeoffs

<table>
<thead>
<tr>
<th>Type(#bits)</th>
<th>Half (16)</th>
<th>Single (32)</th>
<th>Double(64)</th>
<th>Quad(128)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mantissa = P</td>
<td>11</td>
<td>24</td>
<td>53</td>
<td>113</td>
</tr>
<tr>
<td>Bin width = W</td>
<td>7</td>
<td>13</td>
<td>40</td>
<td>100</td>
</tr>
<tr>
<td>[Wmin,Wmax]</td>
<td>[7,8]</td>
<td>[13,21]</td>
<td>[28,50]</td>
<td>[58,110]</td>
</tr>
<tr>
<td>#Bins = K = Kmin</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Accuracy = (K-1)*W</td>
<td>14</td>
<td>26</td>
<td>80</td>
<td>200</td>
</tr>
<tr>
<td>MaxDep = 2^(P-W-2)</td>
<td>4</td>
<td>512</td>
<td>2048</td>
<td>2048</td>
</tr>
<tr>
<td>MaxN = 2^(2P-W-2)</td>
<td>2^13</td>
<td>2^33</td>
<td>2^64</td>
<td>2^124</td>
</tr>
</tbody>
</table>

- Accuracy = #Bits in Indexed Sum = (K-1)*W
- K ≥ Kmin = 3 => Accuracy ≥ P
- MaxDep = #Deposits before Renorm = 2^(P-W-2) (so Wmax = P-3)
- MaxN = max #summands in worst case = 2^(2P-W-2)
- Wmin => only top bin requires scaling to avoid overflow
- Tradeoffs: W larger => more Accuracy, lower MaxDep/MaxN
  K larger => more Accuracy, more Flops (3K-2), more memory (2K)
Current status of ReproBLAS (1/2)

• Sequential only, so far; same interface
  – BLAS1: r{s,d}dot, r{z,c}dot{u,c}, r{d,s,c,z}sum, r{d,s,dz,sc}{nrm2,asum}
  – BLAS2: r{d,s,z,c}gemv
  – BLAS3: r{d,s,z,c}gemm

• MPI Reduce Sum to sum Indexed Types

• Future work
  – Other functions
  – OpenMP, MPI versions
Current status of ReproBLAS (2/2)

• Written in C (C99 conformant)
• Code generation and testing in Python
• Vectorization using Intel AVX or SSE
  – Extensible to AVX-512 in future
  – Vectorization and cache blocking autotuned using OpenTuner
• Doxygen for documentation
Timing Results

• Tuned for Intel Core i7-2600
  – 3.4GHz, 32 KB L1, 256 KB L2, 8MB L3 caches
  – Gcc ver 4.8.4 with –O3 flag

• Each test run ≥ 100 times

• Largest BLAS1 problems fit in L2

• Largest BLAS2 & 3 do not fit in L3

• Comparison vs Intel MKL Ver. 11.0.5

• Theoretical peak (minimum run time) =
  \[
  \frac{\text{max}(\#\text{adds}, \#\text{muls}, \#\text{ors})}{(\text{vector\_register\_size} \times 3.4\text{GHz})}
  \]
Summation

- Compare to gcc –O3 applied to:
  
  ```c
  res=0; for ( j=0; j<N; j++) { res += X[j]; }
  ```

- Reproducible sum faster for large N!
Dot Product

• Compare to MKL ddot, for $N=2^{6:12}$
  – $2^{6:11}$ in L1, $2^{12}$ in L2
  – $3.33 \leq \text{slowdown} \leq 4.15$ for $N \geq 2^{10}$
GEMV

- Compare to MKL dgemv, for $N=2^{[6:12]}$
  - $2^6$ in L1, $2^7$ in L2, $2^{[8:10]}$ in L3, $2^{[11:12]}$ in main mem
  - $5.71 \leq \text{slowdown} \leq 7.70$ for $N \geq 2^{10}$
GEMM

- Compare to MKL dgemm, for $N=2^{[6:12]}$
  - $2^6$ in L2, $2^{[7:9]}$ in L3, $2^{[10:12]}$ in main mem
  - Slowdown $\approx 12.6$ for $N \geq 2^{10}$
Testing

• Reproducibility
  – Input data permuted by reversing, sorting, random shuffling
  – Data summed in blocks of various sizes

• Accuracy
  – See if error bound satisfied for known sums with lots of cancellation
  – See Sec 8.2 in Reproblas paper for details
Relationship to IEEE 754 Standard

• Goal: Accelerate inner loop of Deposit:
  – \( r=x; \) for \( k=1:K-1, \ s = Y_{p_k} + (r|1), \ q=s-Y_{p_k}, \ Y_{p_k} = s, \ r=r-q \)
  – All round-to-nearest-even, \( (r|1) = r \) with bottom bit set to 1

• Basically \((head, tail) = two\_sum(Y_{p_k}, r)\)
  – head = round\((Y_{p_k} + r)\), \ tail = Y_{p_k} + r – head \ exactly
  – \( (r|1) \) makes tie-breaking in rounding independent of \( Y_{p_k} \)
  – Needed for reproducibility

• Others want two-sum, for faster double-double

• Make both happy, with one new operation
  – \((head, tail) = two\_sum(a, b)\)
  – head = \(a+b\), round to nearest, break ties away from 0
  – tail = \(a+b\)-head, exactly

• Under discussion in IEEE 754 Committee
Using fixed point arithmetic

• Based on preliminary experiments by John Hauser
  – Represent each primary/carry field by a 64 bit int
  – Do all operations (Deposit, Renorm etc) using integer and boolean operations

• Possible parameters
  – $K = 2$ bins, $W = 53$ bits $\Rightarrow 1+(K-1)W = 53$ bit accuracy
  – Renorm every 1024 deposits
  – Sum up to $2^{64}$ summands

• Early experiments show 5.2x slowdown vs standard sum –
  – comparison to ReproBLAS – work in progress
Issues to discuss

• Should we just build sequential building blocks, or define parallel version too?
  – Sequential only:
    • Need to output Indexed Types and/or Floats
    • Need to sum vectors and matrices of Indexed Typed
    • Need \text{gemm}(\alpha,A_{\text{float}},B_{\text{float}},C_{\text{indexed}}), so with \beta=1, to support parallel \text{gemm}
  – Parallel too: Which layouts to support?
• Should \( K = \#\text{bins} \) be an input parameter?
  – If we support larger \( K \), should we compute \( x(i) \times y(i) \) more accurately, via fma? XBLAS?
• Do we need all 4 precisions?