ReproBLAS: Reproducible BLAS
http://bebop.cs.berkeley.edu/reproblas/

James Demmel, Nguyen Hong Diep

SC'13 - Denver, CO

Nov 22, 2013
Reproducibility: obtaining bit-wise identical results from different runs of the program on the same input data, regardless of different available resources.

**Cause of nonreproducibility**: *not* by roundoff error but by the *non-determinism* of accumulative roundoff error. Due to the *non-associativity* of floating point addition, accumulative roundoff errors depend on the order of evaluation, and therefore depend on available computing resources.
Sources of non-reproducibility
Sources of non-reproducibility

- number of MPI nodes

Related work:
- Intel MKL 11.0 with CNR: reproducible with fixed number of processors, and fixed data alignment,
- Hardware (NIC) for reproducible reduction operator.
Sources of non-reproducibility

- number of MPI nodes
- MPI reduction tree shape

Related work:
- Intel MKL 11.0 with CNR: reproducible with fixed number of processors, and fixed data alignment,
- Hardware (NIC) for reproducible reduction operator.
Sources of non-reproducibility

- number of MPI nodes
- MPI reduction tree
  shape
- number of cores per
  node

Related work:

- Intel MKL 11.0 with CNR: reproducible with fixed number of processors, and fixed data alignment,
- Hardware (NIC) for reproducible reduction operator.
Sources of non-reproducibility

- number of MPI nodes
- MPI reduction tree shape
- number of cores per node
- data path (1, 2, 4, ...)

Related work:
- Intel MKL 11.0 with CNR: reproducible with fixed number of processors, and fixed data alignment,
- Hardware (NIC) for reproducible reduction operator.
Sources of non-reproducibility

- number of MPI nodes
- MPI reduction tree shape
- number of cores per node
- data path (1, 2, 4, \ldots)
- data ordering

Related work:
- Intel MKL 11.0 with CNR: reproducible with fixed number of processors, and fixed data alignment,
- Hardware (NIC) for reproducible reduction operator.
Sources of non-reproducibility

- number of MPI nodes
- MPI reduction tree shape
- number of cores per node
- data path (1, 2, 4, ...)
- data ordering

Related work:
- Intel MKL 11.0 with CNR: reproducible with fixed number of processors, and fixed data alignment,
Sources of non-reproducibility

- number of MPI nodes
- MPI reduction tree shape
- number of cores per node
- data path (1, 2, 4, …)
- data ordering

Related work:
- Intel MKL 11.0 with CNR: reproducible with fixed number of processors, and fixed data alignment,
- Hardware (NIC) for reproducible reduction operator.
Proposed solution

**Goal**: obtaining *deterministic* roundoff errors.

Exact arithmetic: unacceptably expensive in both long-word arithmetics and communication

Fixed point arithmetic:
- limited in exponent range
- limited in value range

Indexed floating point: ¹
- Use floating point numbers to represent extended-precision fixed point,
- Exponent is of format $i \cdot W$ and is adjusted (on-line) according to the maximum absolute value of input,
- Can add up to $2^{60}$ floating point numbers,
- No extra reduction required for distributed environment,

ReproBLAS: Reproducible Basic Linear Algebra Subprograms

**URL:** http://bebop.cs.berkeley.edu/reproblas

- ReproBlas_mpi
- ReproBlas_seq
- IBlas
- IndexedFP
- MPIndexedFP

**Features:**
- Reproducibility,
- Accuracy: no severe accuracy loss,
- Reasonable arithmetic cost,
- Minimize communication: only one reduction per sum.

**Requirements:** ROUND-TO-NEAREST and no overflow
ReproBLAS: Reproducible Basic Linear Algebra Subprograms

URL: http://bebop.cs.berkeley.edu/reproblas

IndexedFP implement arithmetics for Indexed Floating Point:
- Data types: Idouble, Ifloat, dIcomplex, sIcomplex
- Conversion to: dconv2I, sconv2I, zconv2I, cconv2I
- Conversion from: Iconv2d, Iconv2f, Iconv2z, Iconv2c
- Addition: dIAdd, sIAdd, zIAdd, cIAdd
- Cross-type operations: dIAdddd, sIAdddf, zIAdddz, cIAdddc
ReproBLAS: Reproducible Basic Linear Algebra Subprograms

URL: http://bebop.cs.berkeley.edu/reproblas

MPIndexedFP is an MPI wrapper for IndexedFP:

- Data types: MPI_IDOUBLE, MPI_IFLOAT, MPI_IDOUBLE_COMPLEX, MPI_ICOMPLEX
- Reduction operators: MPI_RSUM, MPI_RNRM2
ReproBLAS: Reproducible Basic Linear Algebra Subprograms

URL: http://bebop.cs.berkeley.edu/reproblas

IBlas (Indexed Blas) provides performance-optimized kernel routines which take in vectors in native data type and return result in Indexed Floating Point format

- \( \{s|d|c|z\} \)asumI
- \( \{s|d|c|z\} \)sumI
- \( \{s|d|c|z\} \)nrm2I
- \( \{s|d|c|z\} \)dot{c|u}I
ReproBLAS: Reproducible Basic Linear Algebra Subprograms

URL: http://bebop.cs.berkeley.edu/reproblas

Current version only supports level 1 routines for 4 basic data types:

- \{r|pr}\{s|d|c|z\} asum
- \{r|pr}\{s|d|c|z\} sum
- \{r|pr}\{s|d|c|z\} nrm2
- \{r|pr}\{s|d|c|z\} dot \{c|u\}
Example 1: sum of sines

// Compute sum of $\sin(2\cdot M\_PI\cdot (i/(\text{double})n-0.5))$

double sumsin(int n) {
    int i;
    double t;
    double s;

    s = 0.0;

    for (i = 0; i < n; i++) {
        t = sin(2* M\_PI* (i/(\text{double})n-0.5));
        s = s + t;
    }

    return s;
}
Example 1: sum of sines (Reproducible)

```c
#include <IndexedFP.h>

double sumsin(int n) {
    int i;
    double t;
    double s;   // declare an indexed fp

    s = 0.0;

    for ( i = 0; i < n; i++) {
        t = sin(2*M_PI*(i/(double)n-0.5));
        s = s + t;
    }

    return s;
}
```
Example 1: sum of sines (Reproducible)

```c
#include <IndexedFP.h>

double sumsin(int n) {
    int i;
    double t;
    double s;  // declare an indexed fp
    dISetZero(s);  // Initialize to zero

    for (i = 0; i < n; i++) {
        t = sin(2*M_PI*(i/(double)n-0.5));
        s = s + t;
    }

    return s;
}
```
Example 1: sum of sines (Reproducible)

```c
#include <IndexedFP.h>

double sumsin(int n) {
    int i;
    double t;
    double s; // declare an indexed fp
    dISetZero(s); // Initialize to zero

    for (i = 0; i < n; i++) {
        t = sin(2* M_PI *(i/(double )n -0.5));
        dIAddd(&s,t); // Aggregation
    }

    return s;
}
```
#include <IndexedFP.h>

double sumsin(int n) {
    int i;
    double t;
    IndexedFP s; // declare an indexed fp

    dISetZero(s); // Initialize to zero

    for (i = 0; i < n; i++) {
        t = sin(2*M_PI*(i/(double)n-0.5));
        dIAddd(&s,t); // Aggregation
    }

    return Iconv2d(s); // convert back to normal FP
}
Example 1: sum of sines (Parallel Reproducible)

```c
#include <MPIndexedFP.h>

double sumsin(int n) {
    int i;
    double t;
    double s, s1;  // declare an indexed fp

    /* PARTITIONING. Local work: from start to end */

dISetZero(s);  // Initialize to zero
for ( i = start; i < end; i++ ) {
    t = sin(2 * M_PI * (i / (double) n - 0.5));
    dIAddd(&s, t);  // Aggregation
}

RMPI_Init();  // Initialize Reproducible MPI
MPI_Reduce(&s, &s1, 1, MPI_IDOUBLE, MPI_RSUM, 0, MPI_COMM_WORLD);

return Iconv2d(s1);  // convert back to normal FP
}
```
Example 2: vector summation (naive)

```c
int n = 1000000;
double* v = (double*)malloc(n*sizeof(double));

for (i=0;i<n;i++) v[i]=sin(2*M_PI*(i/(double)n-0.5));

double sum(int n, double* v) {
    int i;
    double t;
    double s;

    s = 0;    // Initialize to zero

    for ( i = 0; i < n; i++)
        s += v[i];

    return s;
}
```
Example 2: vector summation (reproducible)

```c
#include <IndexedFP.h>

int n = 1000000;
double* v = (double*) malloc(n*sizeof(double));

for (i=0;i<n;i++) v[i]=sin(2*M_PI*(i/(double)n-0.5));

double sum_I(int n, double* v) {
    int i;
double t;
    double s; // declare an indexed fp
    dISetZero(s); // Initialize to zero

    for ( i = 0; i < n; i++)
        dIAddd(&s,v[i]); // Aggregation

    return Iconv2d(s); // convert back to normal FP
}
```
#include <rblas.h>

int n = 1000000;
double* v = (double*) malloc(n*sizeof(double));

for (i=0;i<n;i++) v[i]=sin(2*M_PI*(i/(double)n-0.5));

extern double rdsum(int n, double* v, int inc);

double sum_I(int n, double* v) {
    return rdsum(n, v, 1);
}
Example 2: vector summation (parallel reproducible blas)

```c
#include <rblas_mpi.h>

// double* v : local vector
// int n : length of local vector

extern double prdsum(MPI_Comm* com, int root, int n, double* v, int inc);

double sum_I(int n, double* v) {
    return prdsum(MPI_COMM_WORLD, 0, n, v, 1);
}
```
Example 2: vector summation (blocked)

```c
#include <IndexedFP.h>
#define NB 1024
int n = 1000000;
// v[i] = \sin(2*\text{M\_PI}*(i/(\text{double})n-0.5)

double sum_I(int n, double * v) {
  int i, LN;
  double t;
  double s; // declare an indexed fp
  dISetZero(s); // Initialize to zero
  for (i = 0; i < n; i += NB, v += NB) {
    LN = NB < (n-i) ? NB : (n-i);
    dIAAddd(&s, sum(LN, v)); // Block Aggregation
  }
  return Iconv2d(s); // convert back to normal FP
}
```
#include <rblas.h>

#define NB 128

double sumsin(int n) {
    int i, j;
    double t;
    double s, s1;  // declare an indexed fp
    double BUFFER[NB];

    dISetZero(s);  // Initialize to zero

    for (i = 0; i < n; i+=NB) {
        lN = NB < (n-i) ? NB : (n-i);
        for (j = 0; j < lN; j++)
            BUFFER[j] = sin(2*M_PI*((i + j)/(double)n-0.5));
        s1 = dsumI(lN, BUFFER, 1);  // Indexed BLAS call
        dIAdd(&s,s1);  // Aggregation
    }

    return Iconv2d(s);  // convert back to normal FP
}
Development status

http://bebop.cs.berkeley.edu/reproblas

X: complete  O: in progress

<table>
<thead>
<tr>
<th></th>
<th>Single-threaded</th>
<th>Multi-threaded</th>
<th>GPU</th>
<th>MPI</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Scalar</td>
<td>SSE</td>
<td>AVX</td>
<td>OMP</td>
<td>pthread</td>
</tr>
<tr>
<td>BLAS 1</td>
<td>asum</td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>sum</td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>dot</td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>nrm2</td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>2</td>
<td>gemv</td>
<td>O</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>trsv</td>
<td>O</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>gemm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>trsm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Future work: LAPACK, hardware support, non-linear operations, ...