ReproBLAS: Reproducible BLAS

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

James Demmel, **Nguyen Hong Diep**

SC'13 - Denver, CO

Nov 22, 2013

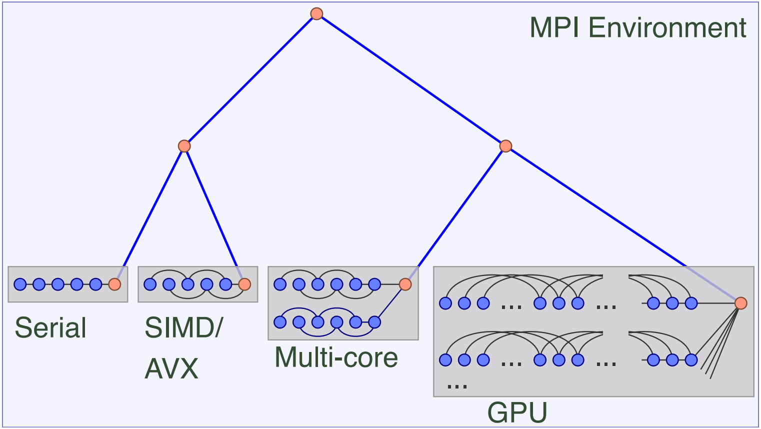
Reproducibility

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

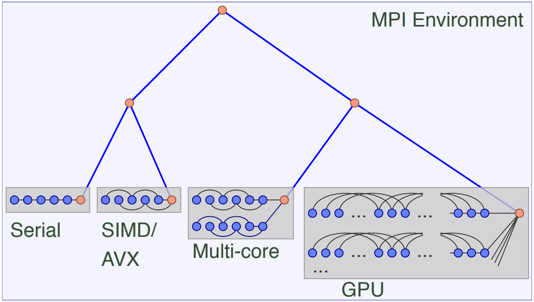
Cause of nonreproducibility: *not* by roundoff error but by the *non-determinism* of accumulative roundoff error.

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

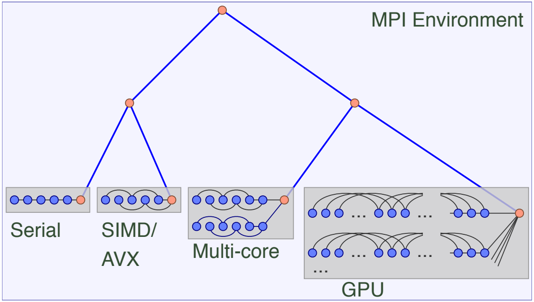


Sources of non-reproducibility



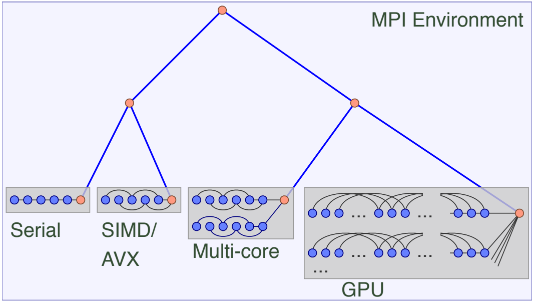
- number of MPI nodes

Sources of non-reproducibility



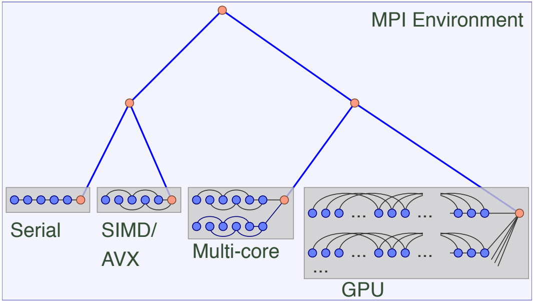
- number of MPI nodes
- MPI reduction tree shape

Sources of non-reproducibility



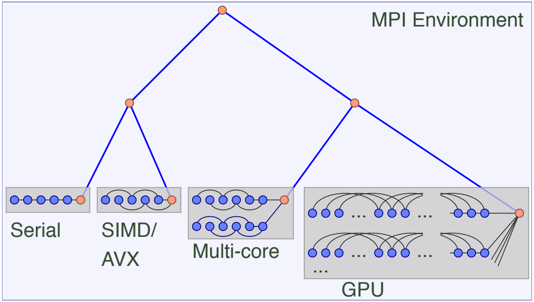
- number of MPI nodes
- MPI reduction tree shape
- number of cores per node

Sources of non-reproducibility



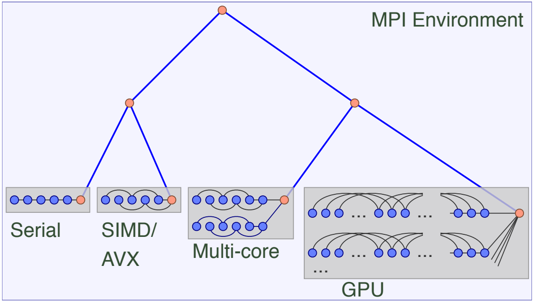
- number of MPI nodes
- MPI reduction tree shape
- number of cores per node
- data path (1,2,4,. . .)

Sources of non-reproducibility



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

Sources of non-reproducibility

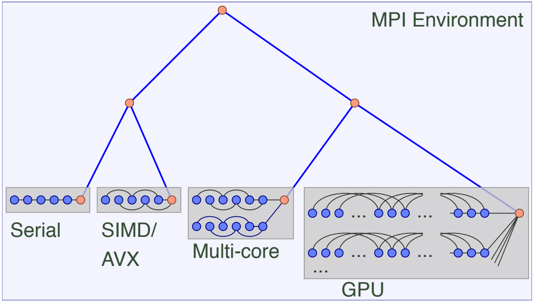


- number of MPI nodes
- MPI reduction tree shape
- number of cores per node
- data path (1,2,4,. . .)
- data ordering

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

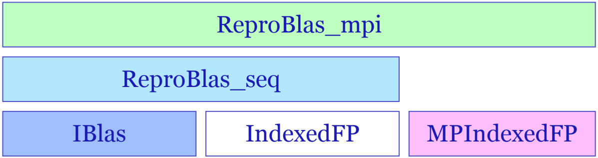
Indexed floating point: [1]

- 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,

[1] J. Demmel and H. D. Nguyen, *Fast Reproducible Floating-Point Summation*, ARITH21, Austin, Texas, April 7–10, 2013

ReproBLAS: Reproducible Basic Linear Algebra Subprograms

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



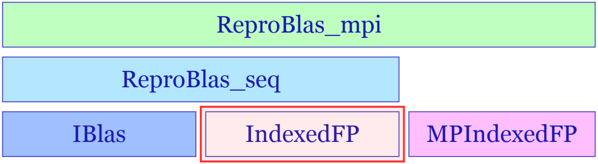
Features:

- 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

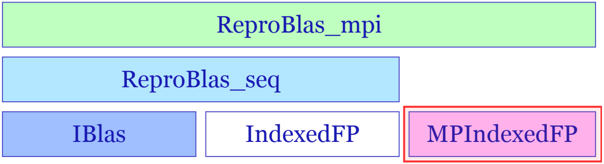


`IndexedFP` implement arithmetics for Indexed Floating Point:

- 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: `dIAddd`, `sIAddf`, `zIAddz`, `cIAddc`

ReproBLAS: Reproducible Basic Linear Algebra Subprograms

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

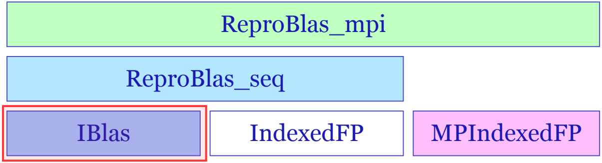


`MPIndexedFP` is an MPI wrapper for `IndexedFP`:

- 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

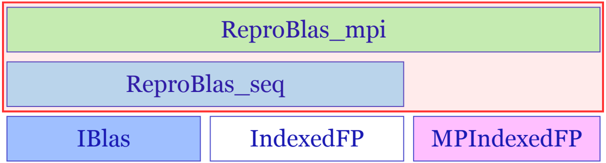


`IBlas` (Indexed Blas) provides peformance-optimized kernel routines which take in vectors in native data type and return result in Indexed Floating Point format

- `{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*M_PI*(i/(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/(double)n-0.5));
      s = s + t;
   }

   return s;
}
```

Example 1: sum of sines (Reproducible)

```
#include <IndexedFP.h>

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

```
#include <IndexedFP.h>

double sumsin(int n) {
   int i;
   double  t;
   Idouble 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;
    Idouble 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;
}
```

Example 1: sum of sines (Reproducible)

```
#include <IndexedFP.h>

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

```
#include <MPIndexedFP.h>

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

```
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;
   Idouble 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
}
```

Example 2: vector summation (reproducible blas)

```
#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)

```
#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)

```
#include <IndexedFP.h>
#define NB 1024
int n = 1000000;
// v[i] = sin(2*M_PI*(i/(double)n-0.5)

double sum_I(int n, double* v) {
   int i, LN;
   double  t;
   Idouble 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);
      dIAddd(&s,sum(LN,v)); // Block Aggregation
   }
   return Iconv2d(s);  // convert back to normal FP
}
```

Example 1: sum of sines (Revisited)

```c
#include <rblas.h>

#define NB 128
double sumsin(int n) {
   int i, j;
   double  t;
   Idouble 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

		Single-threaded			Multi-threaded		GPU	MPI	...
		Scalar	SSE	AVX	OMP	pthread			
BLAS 1	asum	x	x	o	o			x	
	sum	x	x	o	o			x	
	dot	x	x	o	o			x	
	nrm2	x	x	o	o			x	
2	gemv	o	o						
	trsv	o	o						
3	gemm								
	trsm								

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