Reproducibility for Parallel Programming

Robert W. Robey

Eulerian Application Group

Los Alamos National Laboratory

Supercomputing 2013

LA-UR-13-24380

Collaborators

In Search of Numerical Consistency in Parallel Programming, Robey, R.W., Robey, J.M., Aulwes, R., Parallel Computing, Vol. 37, Issues 4-5, April-May 2011, pgs. 217-229

- Jon Robey, UC Davis
- Rob Aulwes, CCS-7, LANL

- **Other helpful sources**
  - M. Cleveland, T. Brunner, N. Gentile, and J. Keasler, Lawrence Livermore National Laboratory (LLNL), Obtaining Identical Results with Double Precision Global Accuracy on Different Numbers of Processors in Parallel Particle Monte Carlo Simulations., Journal of Computational Physics, Vol. 251, Oct 15, 2013
  - D. Bailey, et.al – higher precision arithmetic libraries

Associated exercises are at http://www.github.com/losalamos/ExascaleDocs/Sapient_EPSum.tgz

Reproducibility Problem for Parallel Processing answers change with number of processors

- Finite precision addition not associative
- Order of operations change with number of processors
- Precision errors same order of magnitude as programming errors
  - boundary condition errors
  - old ghost cell values
- Solver iterations change with number of processors
- Stability of method is posited on conservation laws, but cannot check total mass and energy with global sum with enough precision to verify correct implementation

Error of Our Ways

The Prevailing Thought for a Decade (or more)

- **Global sums (such as total mass and energy or total residual error) vary with number of processors**
- **It is an order problem because finite precision arithmetic is not associative**
  - Can be fixed by sorting, but this is too expensive and difficult with distributed memory
- **Solver iterations should use max residual error because total residual error varies with number of processors**

So we just have to live with it …?...

The Revelation

- **It is a precision problem**
  - Enough precision and addition is associative

But Now the Questions

- **How much precision is enough?**
- **What is the impact at the application level?**
- **What is the cost of the precision?**

The Study

- Take a simple compressible fluid dynamics hydrocode with total mass and energy checks
- Compile and run with standard global sums, recording change in total mass and energy
- Besides the order changing of global sums, there is also a time-marching error – which dominates?
- Pure order change in sum is examined by reversing the loops so they run from nsize to 0 instead of 0 to nsize

Compile and Run Sapient

- Sapient Compressible Fluid Dynamics code, courtesy J. Robey and D. Shlachter
- Exercises are at http://www.github.com/losalamos/ExascaleDocs/Sapient_EPSum.tgz
- Untar Sapient_EPSum.tgz with tar –xzvf Sapient_EPSum.tgz
- cmake .
- make
- Run with mpi – mpirun –np 1 Sapient
- 8 Processors – mpirun –np 8 Sapient

How do the errors differ between runs?

Note: The code currently prints out several sums besides the main one for reference purposes.

Using the long data type

- Edit sums.c at line 19 and change to long double type for sum accumulation

```
long double ldsum=0.0;
for(unsigned int j=nbound; j<mysize+nbound; j++){
  for(unsigned int i=nbound; i<isize+nbound; i++){
    ldsum += var[j][i]*deltaX*deltaY;
  }
}
sum = ldsum;
```

The long double type is 80 bits instead of 64 bits on some hardware – this gives about 2 digits more accuracy (and takes 128 bits storage)

Rerun the problem on 8 processors – note that the conservation error improves a little

Another way to change the sum is to change the function pointer at line 64 of Sapient.c from sum_var_normal to sum_var_long and line 65 from sum_var_normal_reverse to sum_var_long_reverse.

Long Double Results

- **Run on one processor (./Sapient or mpirun –n 1 Sapient**

  Conservation diffs: (curr-orig) or (reverse-forward)

  Forward : Mass: -1.164153218e-09 Energy -7.241851563e-11

  Reverse : Mass: -9.791619959e-09 Energy -2.135038812e-10

  Long Double

  Forward : Mass: 1.000444172e-11 Energy 1.818989404e-12

  Reverse : Mass: 1.818989404e-12 Energy -2.273736754e-13

- **Parallel MPI on eight processors (mpirun –n 8 Sapient)**

  Conservation diffs: (curr-orig) or (reverse-forward)

  Forward : Mass: 2.340129868e-09 Energy -9.129053069e-11

  Reverse : Mass: 2.107299224e-09 Energy 6.161826605e-11

  Long Double

  Forward : Mass: 3.637978807e-12 Energy -4.547473509e-13

  Reverse : Mass: 0 Energy 0

Note: Forward is relative to starting totals (drift during calculation). Reverse is relative to forward (order difference effect on global sums

Long Double

- Reproducibility is improved with long double (80 bits)
- But
  - Long double is not portable – on some systems it is only 64 bits
  - Still some error at lower processor counts where most initial parallel code testing is done
- Advantages
  - Simple change
  - Minimal run-time impact (but storage size increases to 16 bytes)

Overall this approach falls a little short as a recommended way to improve parallel reproducibility

Local Kahan Sum

- Now lets try the Kahan Sum using two doubles

```
double corrected_next_term, new_sum, sum=0.0, correction=0.0;
for(unsigned int j=nbound; j<mysize+nbound; j++){
  for(unsigned int i=nbound; i<isize+nbound; i++){
     corrected_next_term = var[j][i]*deltaX*deltaY - correction;
     new_sum = sum + corrected_next_term;
     correction = (new_sum - sum) - corrected_next_term;
     sum = new_sum;
  }   }
```

This gives us essentially 128 bits precision for the local sums with the MPI sums still just a single double.

Compile and rerun – the conservation error is now zero.

Turning on all enhanced precision sum comparisons

- **Edit sums.h**
  - Uncomment line 4 -- #define COMPARE_SUMS 1

- **Several of the enhanced precision sums will print out for reference along with a summary table at the end of different error measures**

Summary Table
1 processor results

```
Conservation max absolute diffs:
        Normal:           Mass:  8.149072528e-09 Energy -5.820766091e-10
        Reverse:          Mass:  2.771648724e-07 Energy                0
        Long:             Mass:  1.000444172e-11 Energy                0
        Long Reverse:     Mass: -4.365574569e-11 Energy                0
        Kahan simple:     Mass:               0 Energy                0
        Kahan corrected:  Mass:               0 Energy                0
        Kahan mpiop:      Mass:               0 Energy                0
        Rev Kahan mpiop:  Mass:               0 Energy                0
Conservation max relative diffs:
        Normal:           Mass:  1.491473611e-12 Energy -1.067479438e-12
        Reverse:          Mass:   5.07280884e-11 Energy -2.584932268e-11
        Long:             Mass:  1.776356839e-15 Energy                0
        Long Reverse:     Mass: -7.993605777e-15 Energy                0
        Kahan simple:     Mass:               0 Energy                0
        Kahan corrected:  Mass:               0 Energy                0
        Kahan mpiop:      Mass:               0 Energy                0
        Rev Kahan mpiop:  Mass:               0 Energy                0
```

Summary Table (cont)

```
Conservation max relative diffs in machine epsilon (adjusted to half
epsilon):
        Normal:          Mass:             13434 Energy             -9615
        Reverse:         Mass:            456918 Energy           -232830
        Long:            Mass:                16 Energy                 0
        Long Reverse:    Mass:               -72 Energy                 0
        Kahan simple:    Mass:                 0 Energy                 0
        Kahan corrected: Mass:                 0 Energy                 0
        Kahan mpiop:     Mass:                 0 Energy                 0
        Rev Kahan mpiop: Mass:                 0 Energy                 0
Conservation max relative diffs calculated from absolute error/orig total:
        Normal:          Mass:   1.49148489e-12 Energy -1.067482042e-12
        Reverse:         Mass:  5.072813106e-11 Energy                 0
        Long:            Mass:  1.831064039e-15 Energy                 0
        Long Reverse:    Mass: -7.990097625e-15 Energy                 0
        Kahan simple:    Mass:                 0 Energy                 0
        Kahan corrected: Mass:                 0 Energy                 0
        Kahan mpiop:     Mass:                 0 Energy                 0
        Rev Kahan mpiop: Mass:                 0 Energy                 0
        MACHINE EPS IS  2.220446049e-16
```

MPI Kahan Sum

- With many processors, we may need to maintain precision in the MPI Sum. To do so, we define a new MPI op as follows:

```
MPI_Type_contiguous(2, MPI_DOUBLE, &MPI_TWO_DOUBLES);
MPI_Type_commit(&MPI_TWO_DOUBLES);

MPI_Op_create((MPI_User_function *)kahan_sum, 1, &KAHAN_SUM);

MPI_Allreduce(&local, &global, 1, MPI_TWO_DOUBLES, KAHAN_SUM, MPI_COMM_WORLD);

MPI_Op_free(&KAHAN_SUM);
MPI_Type_free(&MPI_TWO_DOUBLES);
```

The type and op can be created once at startup and destroyed once at the end of the program rather than every use. The user op is on the next page.

Kahan Sum User Op

```
void kahan_sum( struct esum_type * in , struct esum_type * inout , int *
      len , MPI_Datatype *MPI_TWO_DOUBLES)
{
  double corrected_next_term , new_sum;
  corrected_next_term = in->sum + ( in->correction + inout->correction) ;
  new_sum = inout->sum + corrected_next_term ;
  inout->correction = corrected_next_term - (new_sum – inout->sum) ;
  inout->sum = new_sum;
}
```

Additional Exercises

- **Increasing the number of cells will make the global sum error worse**
  - Each cell's contribution to the global sum will be smaller making the lost precision greater
  - Change line 161 of Sapient.c from 1280 x 1280 to 2048 x 2048 and compare error
- **Change problem from SHOCK_HELL_Y at line 9 of Sapient.h to SHOCK_HELL_X**
  - This will stress the MPI global sum more by increasing the dynamic range of the local sums across processors. The dynamic range within a processor should be lower, reducing the error within a processor
- **Try the the shock in a box problem by changing the problem to SHOCK_IN_BOX in Sapient.h.**
  - This will give results more like a typical problem

Our Discoveries

- **How much precision is enough?**
  - Varies by the problem, but 2 extra digits is not enough (long double or 80 bit numeric registers). Four extra digits is a minimum. Two doubles suffices for most problems. A rounding routine can help with a consistent result.
- **What is the impact at the application level?**
  - We achieve near perfect reproducibility regardless of number of processors. We can also verify the correct implementation of the conservation equations.
- **What is the cost of the precision?**
  - Coding is simple. Run-time cost is low and when MPI Allreduce is factored in, almost free. Applications that do more frequent global sums may need to be more selective in the use of enhanced precision sums.