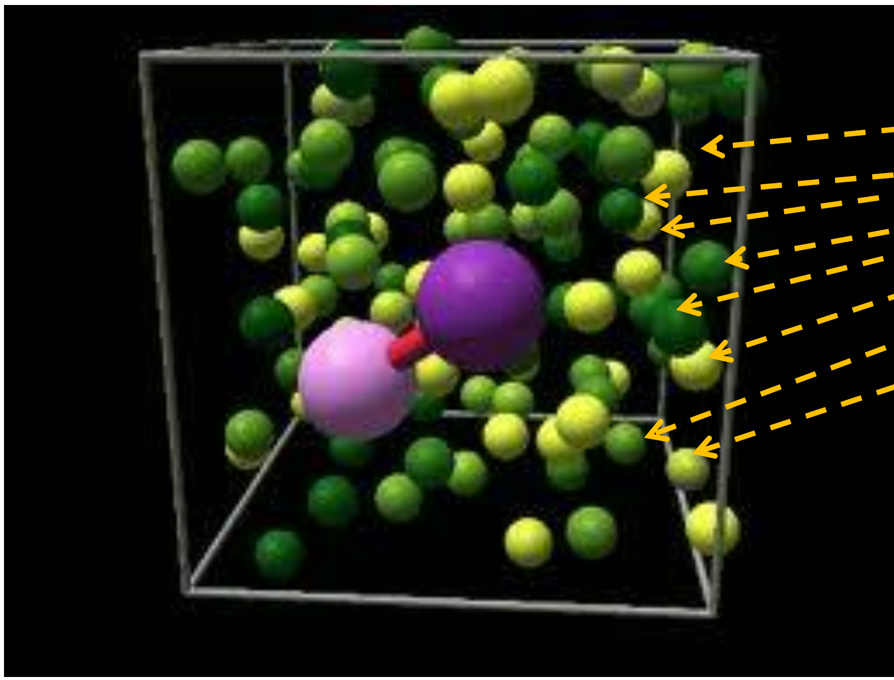
UNIVERSITY of DELAWARE

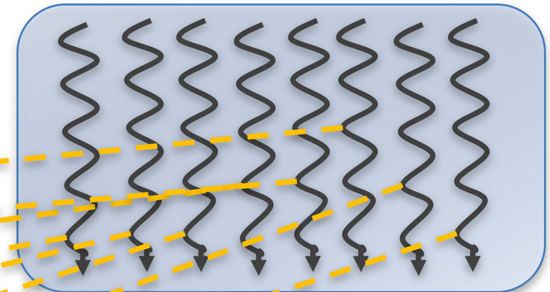
Numerical Reproducibility Challenges on Extreme Scale Multi-Threading GPUs

Dylan Chapp[1], Travis Johnston[1],
Michela Becchi[2], and **Michela Taufer[1]**

[1]University of Delaware
[2]University of Missouri

Molecular Dynamics onto Accelerators





MD simulation step:

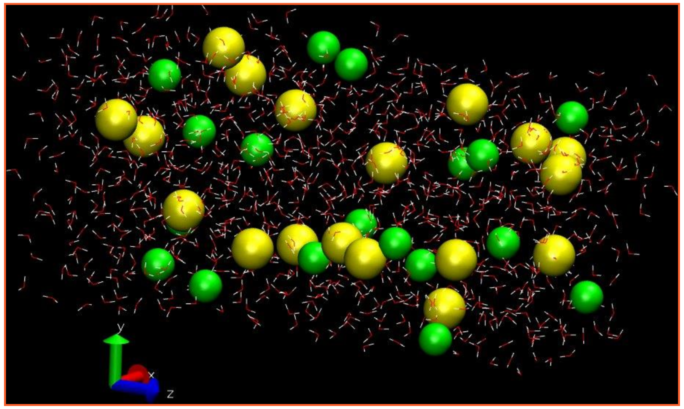
- Each GPU-thread computes forces on single atoms
  - E.g., bond, angle, dihedrals and, nonbond forces
- Forces are added to compute acceleration
- Acceleration is used to update velocities
- Velocities are used to update the positions

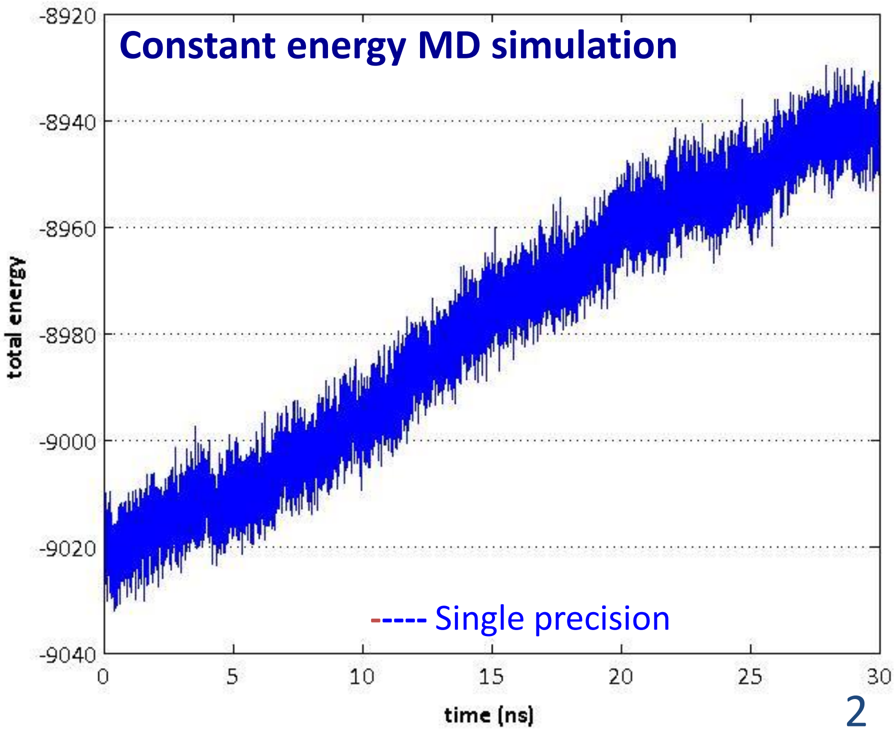
Force -> Acceleration -> Velocity -> Position

The Strange Case of Constant Energy MDs

- Enhancing performance of MD simulations allows simulations of larger time scales and length scales
- GPU computing enables large-scale MD simulation
  - Simulations exhibit unprecedented speed-up factors

MD simulation of NaI solution system containing 988 waters, 18 Na+, and 18 I−: GPU is X15 faster than CPU

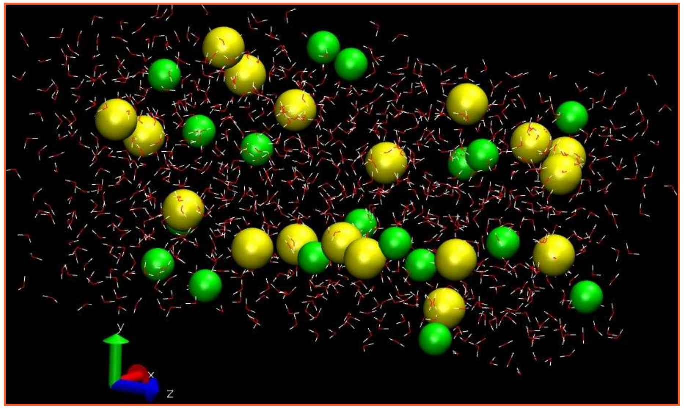


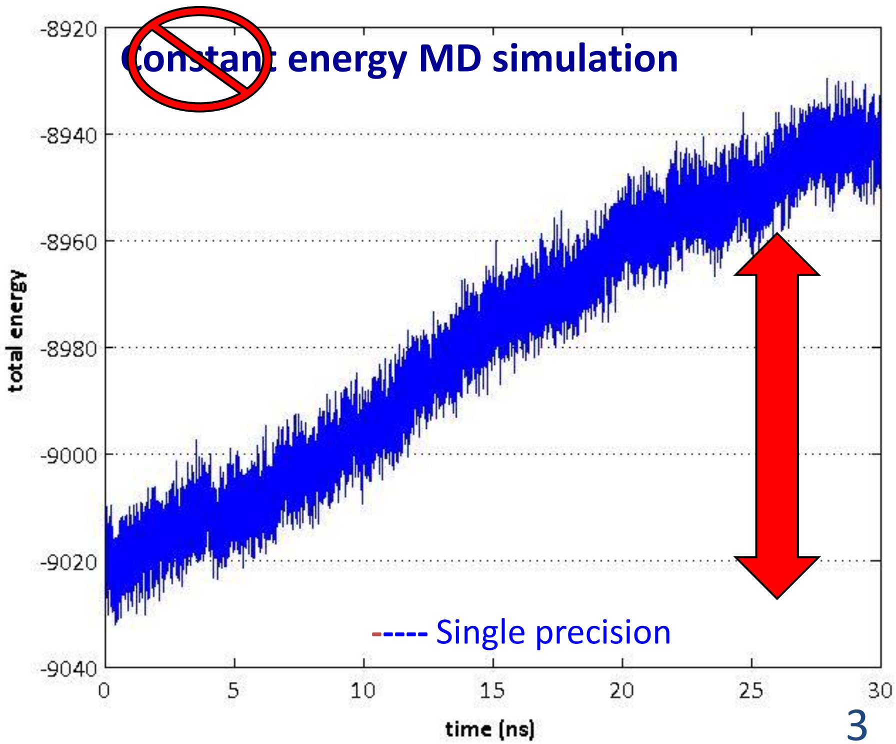


The Strange Case of Constant Energy MDs

- Enhancing performance of MD simulations allows simulations of larger time scales and length scales
- GPU computing enables large-scale MD simulation
  - Simulations exhibit speed-up factors of X10-X30

MD simulation of NaI solution system containing 988 waters, 18 Na+, and 18 I−: GPU is X15 faster than CPU

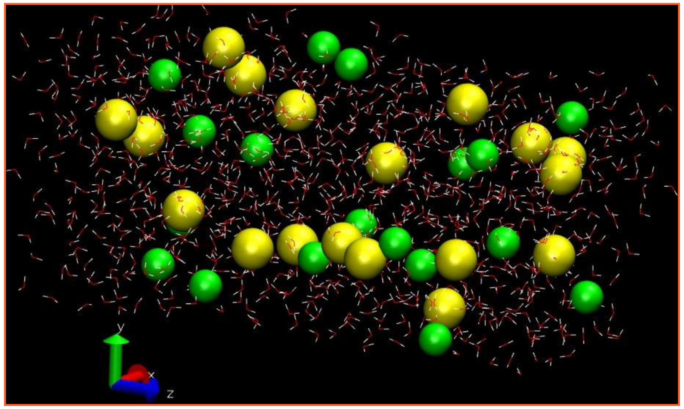


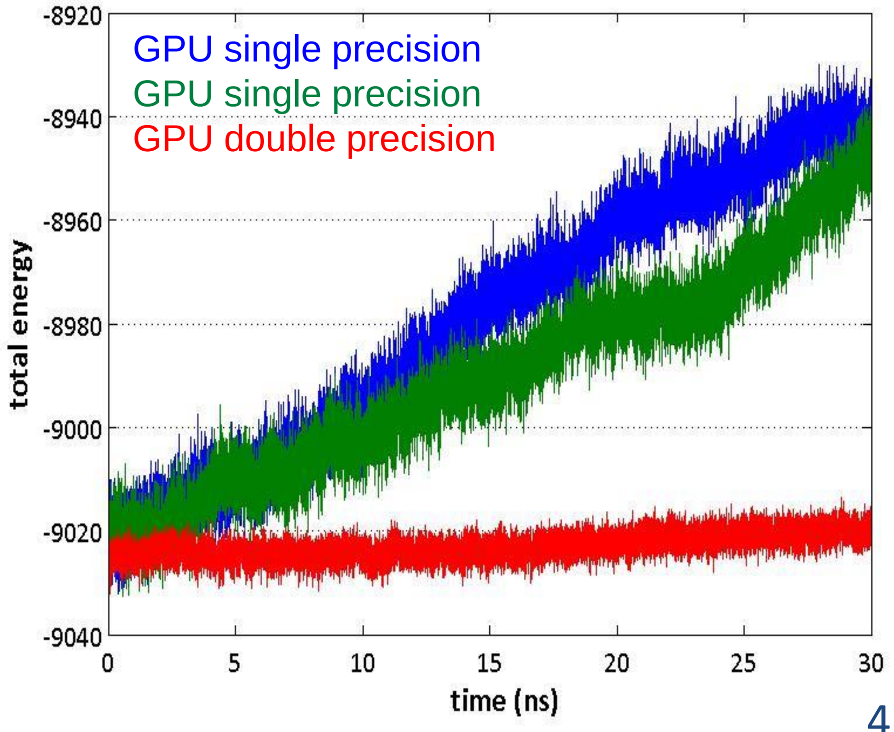


The Strange Case of Constant Energy MDs

- Enhancing performance of MD simulations allows simulations of larger time scales and length scales
- GPU computing enables large-scale MD simulation
  - Simulations exhibit unprecedented speed-up factors

MD simulation of NaI solution system containing 988 waters, 18 Na+, and 18 I−: GPU is X15 faster than CPU

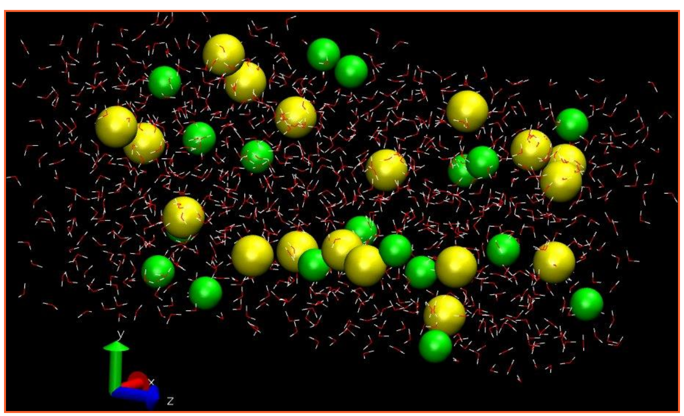


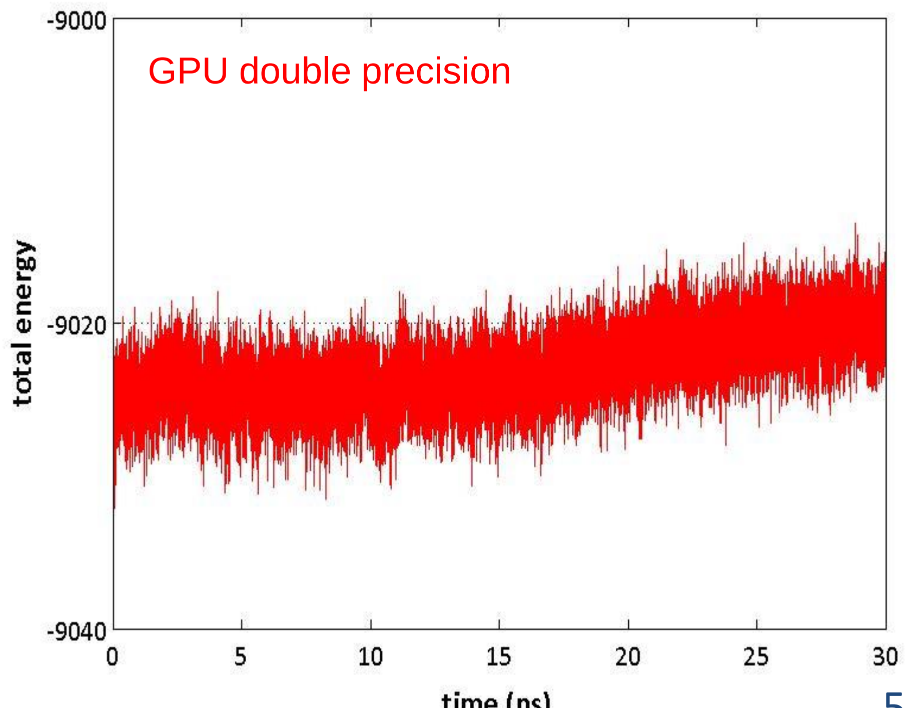


The Strange Case of Constant Energy MDs

- Enhancing performance of MD simulations allows simulations of larger time scales and length scales
- GPU computing enables large-scale MD simulation
  - Simulations exhibit unprecedented speed-up factors

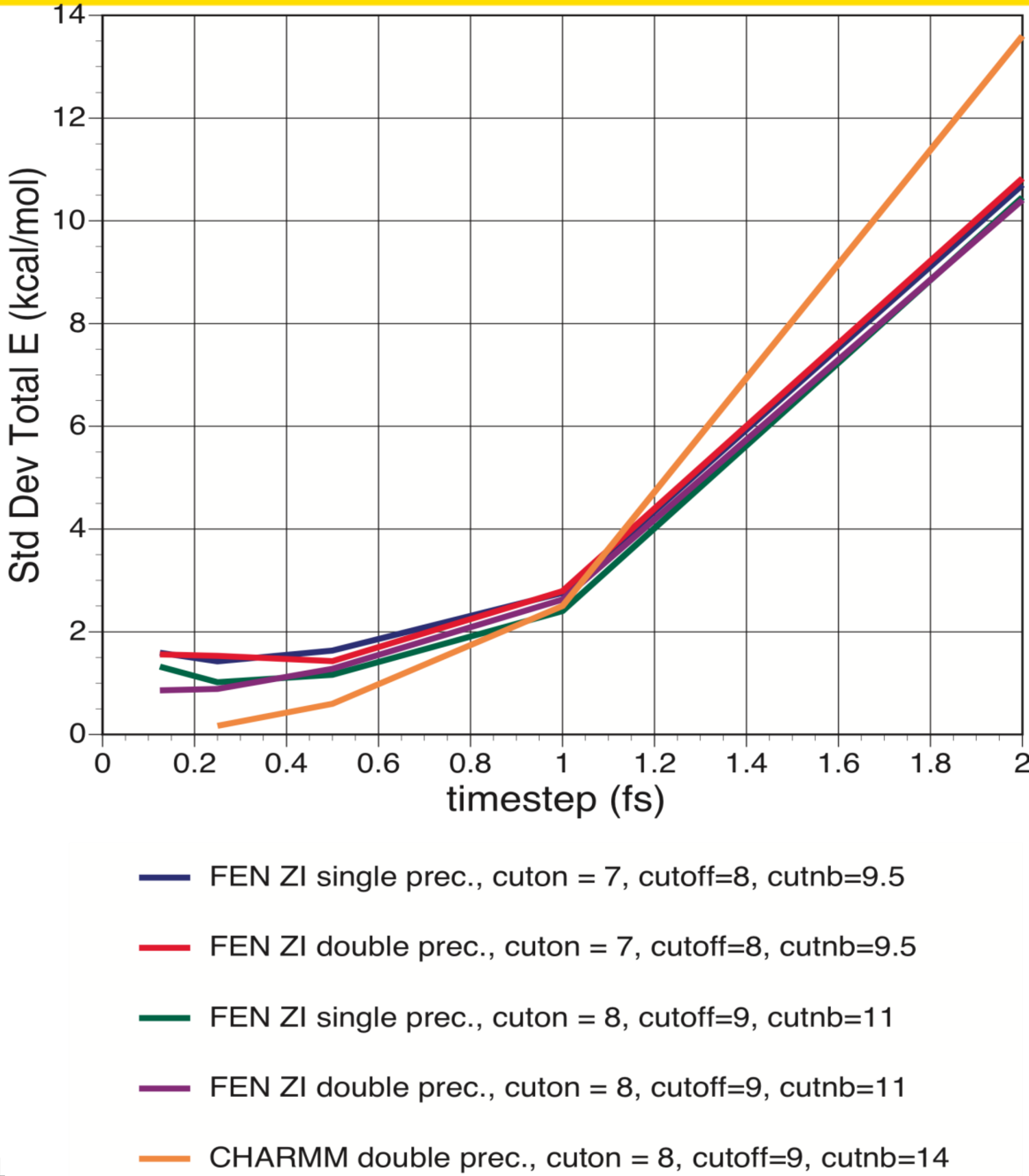
MD simulation of NaI solution system containing 988 waters, 18 Na+, and 18 I−: GPU is X15 faster than CPU





Just a Case of Code Accuracy?

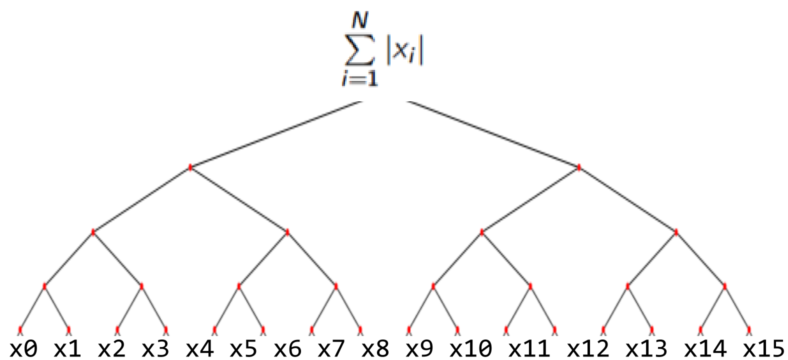
- A plot of the **energy fluctuations versus time step size should follow an approximately logarithmic trend** [1]
- Energy fluctuations are proportional to time step size for large time step size
  - Larger than 0.5 fs
- A different behavior for step size less than 0.5 fs is consistent with results previously presented and discussed in other work [2]



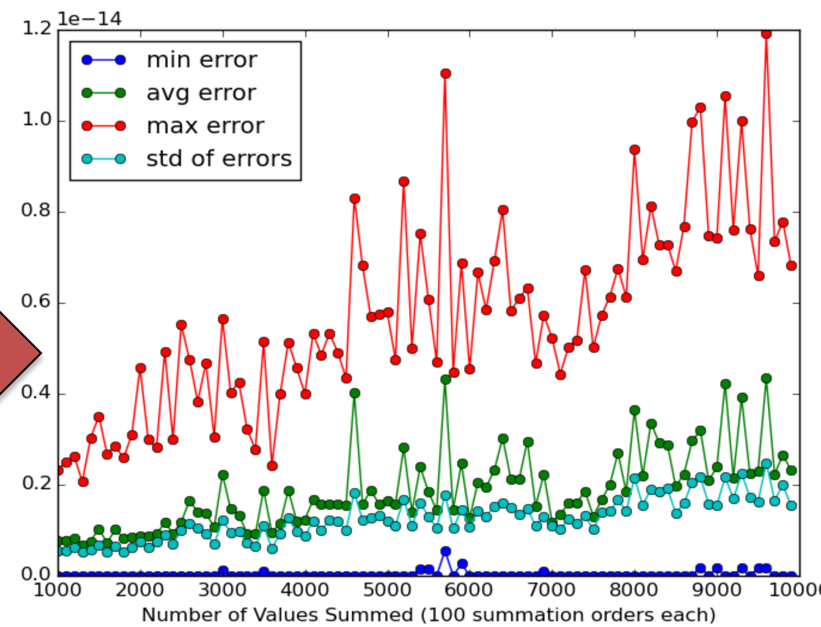
[1] Allen and Tildesley, Oxford: Clarendon Press, (1987)
[2] Bauer et al., J. Comput. Chem. 32(3): 375 – 385, 2011

A Case of Irreproducible Summation

- The modeling of finite-precision arithmetic maps an infinite set of real numbers onto a finite set of machine numbers
- Addition and multiplication of N floating-point numbers is not associative
- **No control** on the way N floating-point numbers are assigned to N threads

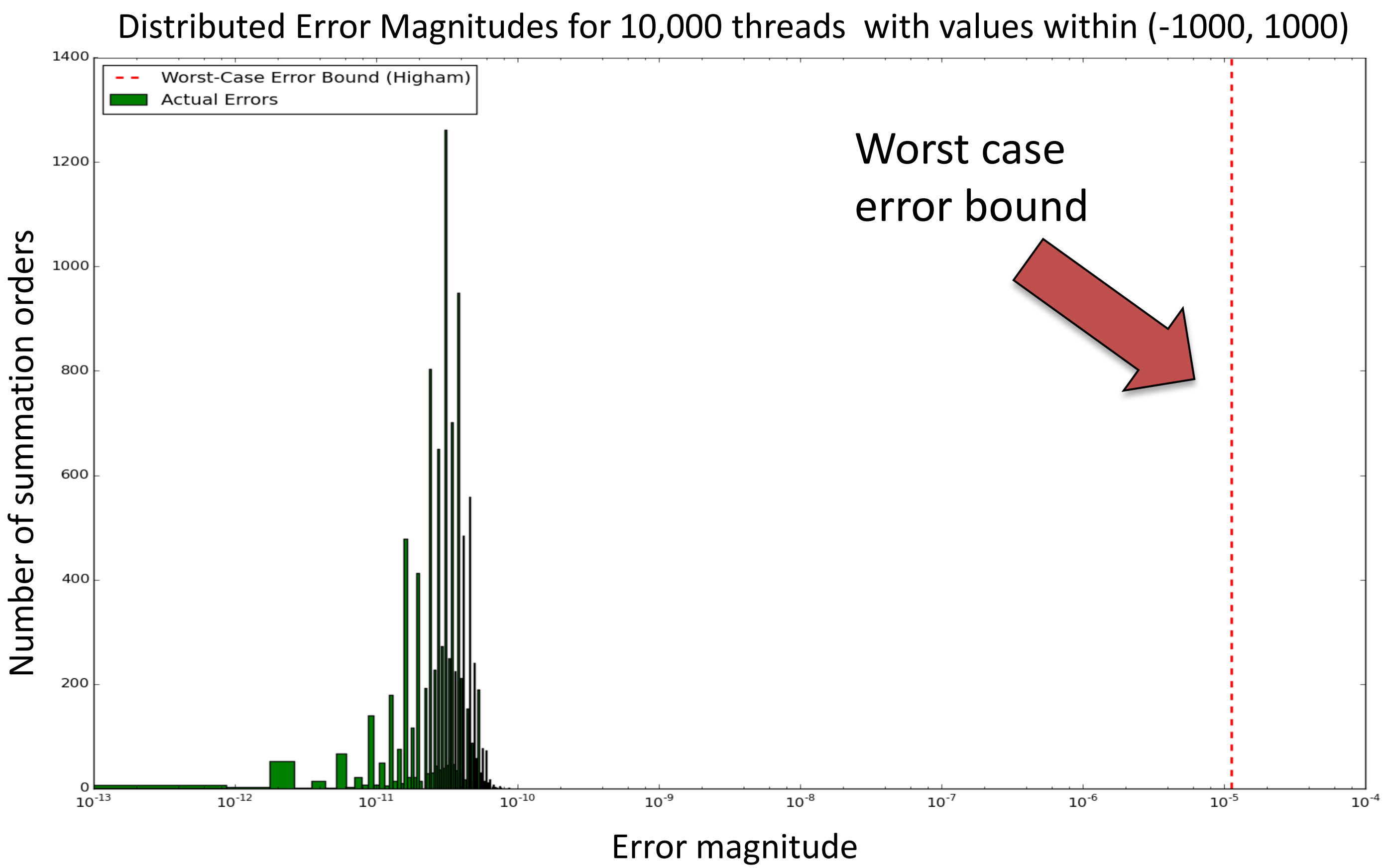


- Different thread orders cause round-off errors to accumulate in different ways, leading to different summation results



Worst-Case Error Bound vs. Actual Errors

- In practice error bounds are overly pessimistic (i.e., usually N * ε << 1) and thus unreliable predictors



Existing Techniques for Increasing Reproducibility of Summation

- Fixed reduction order
  - Ensuring that all floating-point operations are evaluated in the same order from run to run
- Increased precision numerical types
  - Mixed precision - e.g. use of doubles for sensitive computations and floats everywhere else
- Interval arithmetic
  - Replace floating-point types with custom types representing finite-length intervals of real numbers
- Techniques based on error-free transformations
  - Compensated summation e.g., Kahn and composite precision
  - Pre-rounded reproducible summation

Existing Techniques for Increasing Reproducibility of Summation

- Fixed reduction order
  - Ensuring that all floating-point operations are evaluated in the same order from run to run
- Increased precision numerical types
  - Mixed precision - e.g. use of doubles for sensitive computations and floats everywhere else
- Interval arithmetic
  - Replace floating-point types with custom types representing finite-length intervals of real numbers
- Techniques based on error-free transformations
  - Compensated summation e.g., Kahn and composite precision
  - Pre-rounded reproducible summation

TOO COSTLY!!!

Existing Techniques for Increasing Reproducibility of Summation

- Fixed reduction order
  - Ensuring that all floating-point operations are evaluated in the same order from run to run
- Increased precision numerical types
  - Mixed precision - e.g. use of doubles for sensitive computations and floats everywhere else
- Interval arithmetic
  - Replace floating-point types with custom types representing finite-length intervals of real numbers

TOO COSTLY!!!

- Techniques based on error-free transformations
  - **Compensated summation** e.g., Kahn and **composite precision**
  - Pre-rounded reproducible summation

Composite Precision: Data Structure

- Decompose a numeric value into two single precision floating-point numbers: a value and an error

```
struct float2{
    float val;   // Value or result
    float err;   // Error approximation
} x₂;

float2 x2  = x₂.val + x₂.err
```

- Each arithmetic operation takes float2s as parameters and returns float2s
  - Error carried through each operation
  - Operations rely on self-compensation of rounding errors

Composite Precision: Addition

Pseudo-code

float2 x_2, y_2, z_2

$z_2 = x_2 \oplus y_2$

Implementation

float2 x_2, y_2, z_2
float t
Z_2.val = x_2.val + y_2.val
t = z_2.val - x_2.val
Z_2.err = x_2.val - (z_2.val – t) +
(y_2.val – t) + x_2.err + y_2.err

- Mathematically z_2.err should be 0
  - But errors introduced by floating-point operations usually result in z_2.err being non-zero
- Subtraction is the same as addition, but y_2.val = –y_2.val and y_2.err = -y_2.err

Composite Precision: Multiplication and Division

Multiplication

Pseudo-code

float2 x_2, y_2, z_2

$z_2 = x_2 * y_2$

Implementation

```
float2  x2, y2, z2
Z2.val = x2.val  * y2.val
Z2.err = (x2.val * y2.err) +
         (x2.err * y2.val) +
         (x2.err * y2.err)
```

Division

Pseudo-code

float2 x_2, y_2, z_2

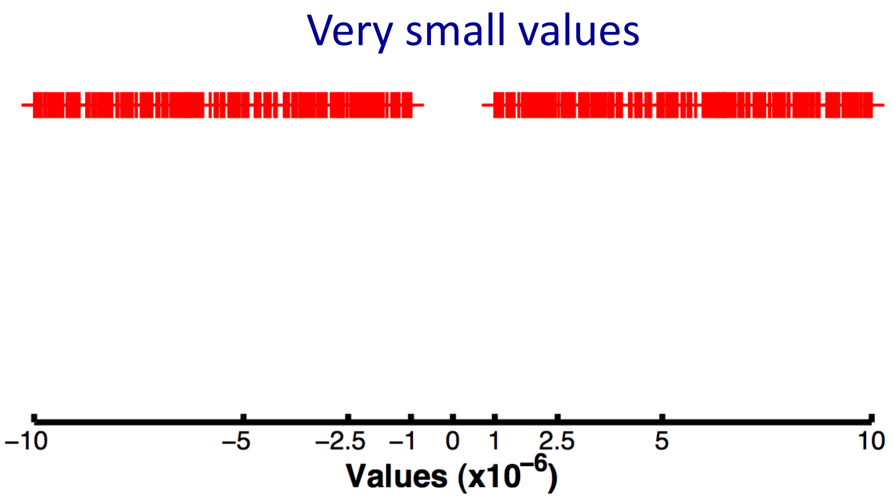
$z_2 = x_2 / y_2$

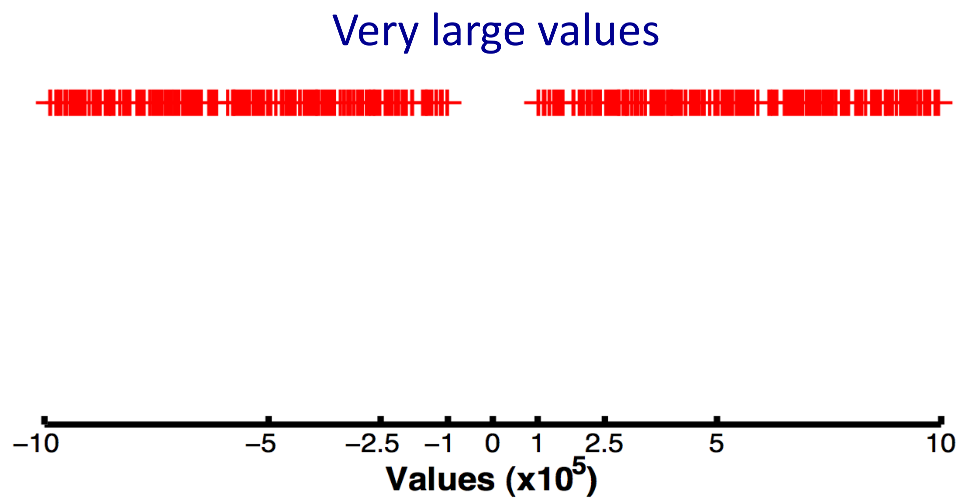
Implementation

```
float2  x2, y2, z2
float t, s, diff
t = (1 / y2.val)
s = t * x2.val
diff = x2.val - (s * y2.val )
Z2.val  = s
Z2.err  = t * diff
```

Global Summation

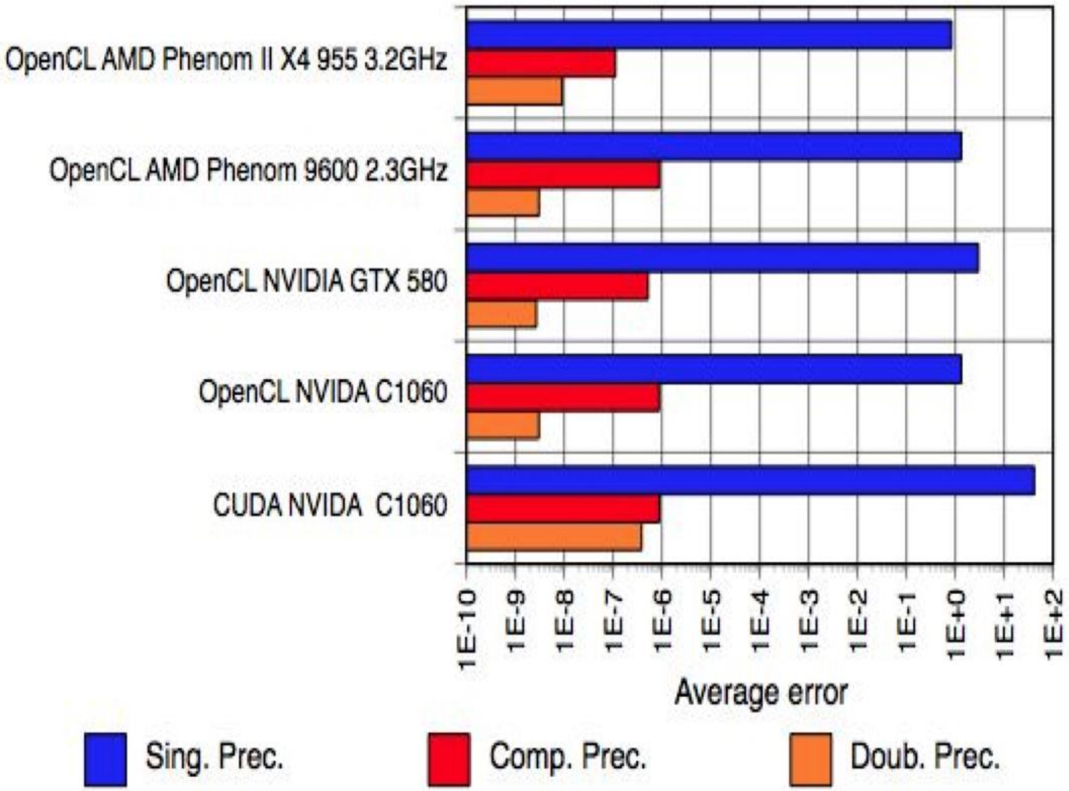
- Randomly generate an array filled with very large – e.g., $O(10^6)$ - and very small – e.g., $O(10^{-6})$ - numbers
  - Whenever you generate a number, the next number should be its negative
  - The total sum *should* be 0





Pre-Fermi GPUs Era

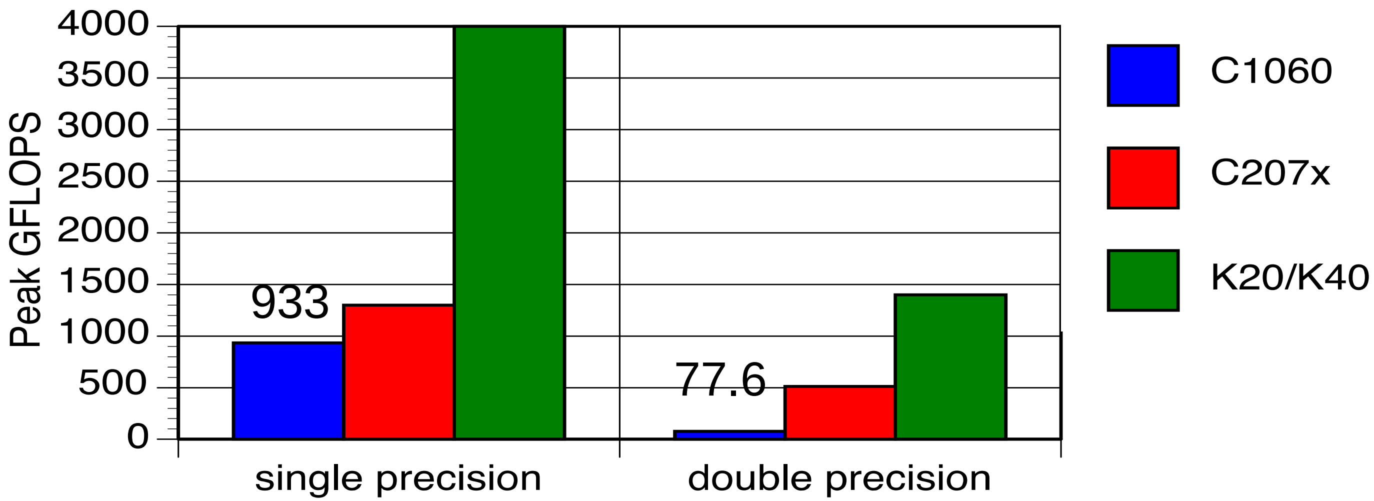
- Randomly shuffled array of 1,000 values on a broad range of multi-core platforms
- Accuracy:
  - Double precision error is very small (10^{-8} to 10^{-9})
  - Single precision error is large (10^{+0})
  - Comp. prec. errors is close to the double precision (10^{-6} to 10^{-7})
- Performance:
  - Double precision is 10 times larger than single precision



[1] *Taufer et al. IPDPS (2010)*

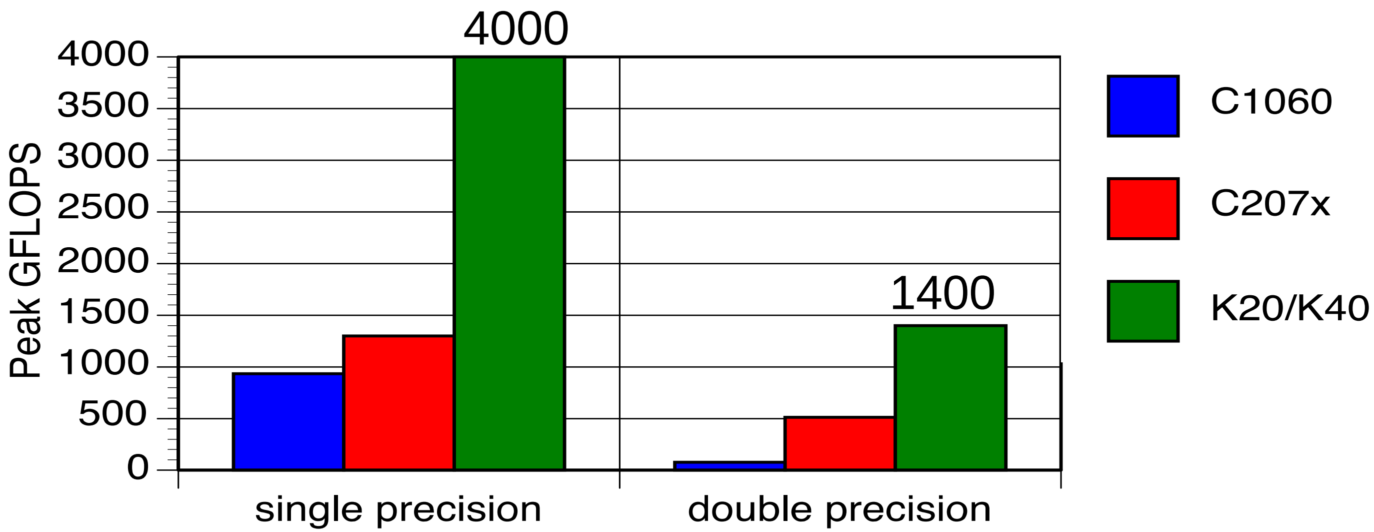
From the pre-Fermi to the Fermi GPUs Era

- On pre-Fermi GPUs, composite precision was a good compromise between result accuracy and performance
  - The performance slow-down of double precision arithmetic was 10 times that of single precision arithmetic



From the pre-Fermi to the Fermi GPUs Era

- On pre-Fermi GPUs, composite precision was a good compromise between result accuracy and performance
  - The performance slow-down of double precision arithmetic was 10 times that of single precision arithmetic
- On Fermi GPUs, the difference in performance between the two has significantly decreased

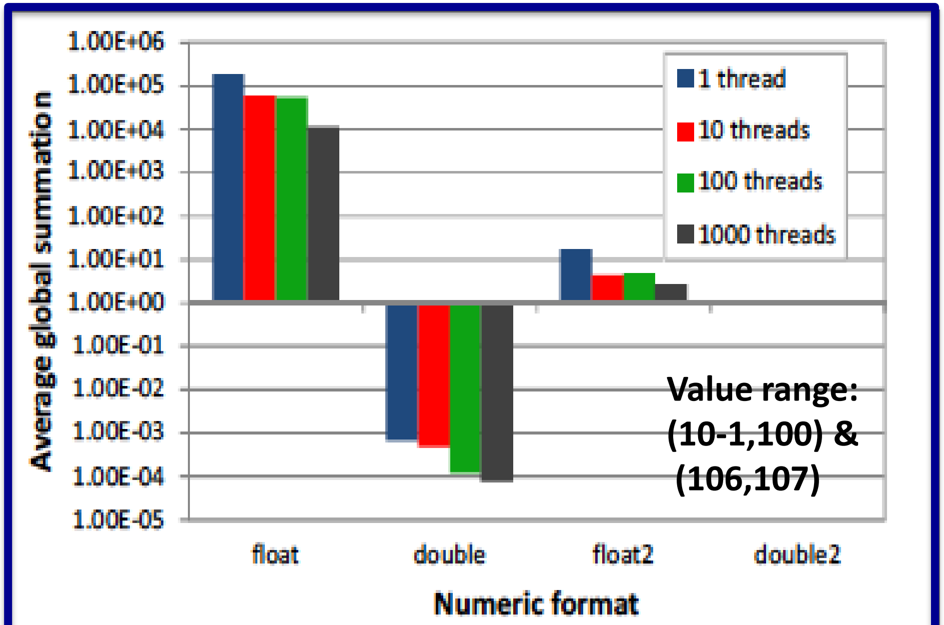


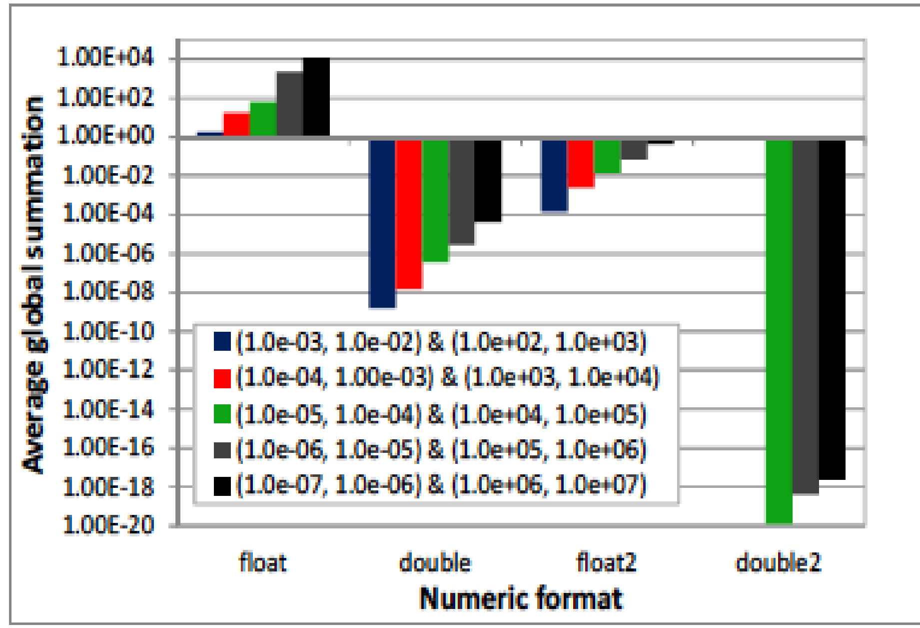
Newly Explored Space

- We perform experiments on more recent **Kepler GPUs** as well as **multi-core CPUs** and **Intel Phi** coprocessor devices
- We consider single, double, and composite precision (both float2 and double2) arithmetic
- We test larger datasets (up to 10 million elements)
- We study different work partitioning and thread scheduling schemes
- We test existing multiple precision floating point libraries (i.e., GNU Multiple Precision Library on multicore CPUs and CUMP on GPUs)

Accuracy on Kepler GPUs

Bars represent average absolute values of global summation over 4 runs
The expected result is 0: the smaller value, the better accuracy



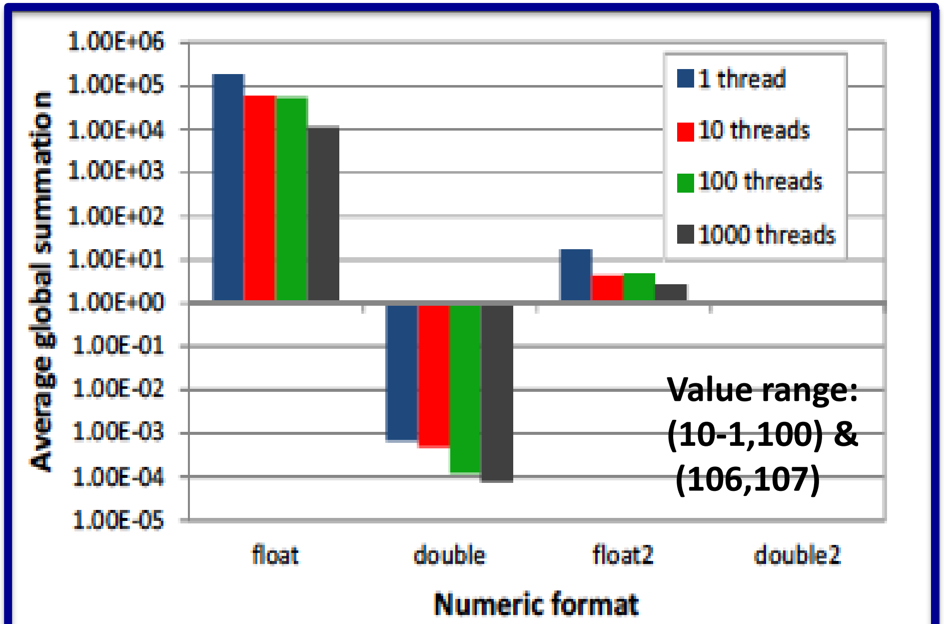


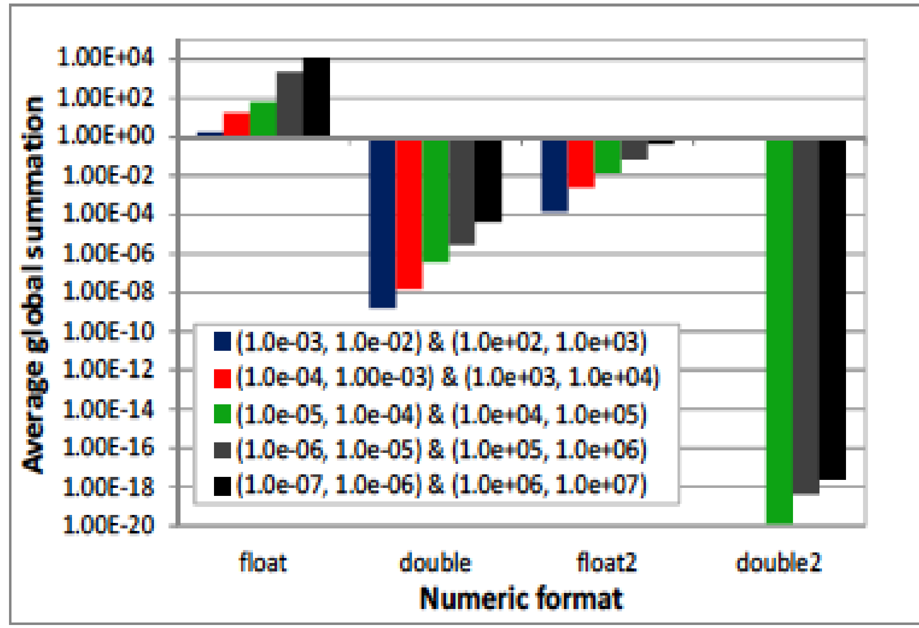
Single precision arithmetic (float) leads to a significant result drift: the computed global summation is as high as 100,000!

UNIVERSITY *of* DELAWARE

Accuracy on Kepler GPUs

Bars represent average absolute values of global summation over 4 runs
The expected result is 0: the smaller value, the better accuracy

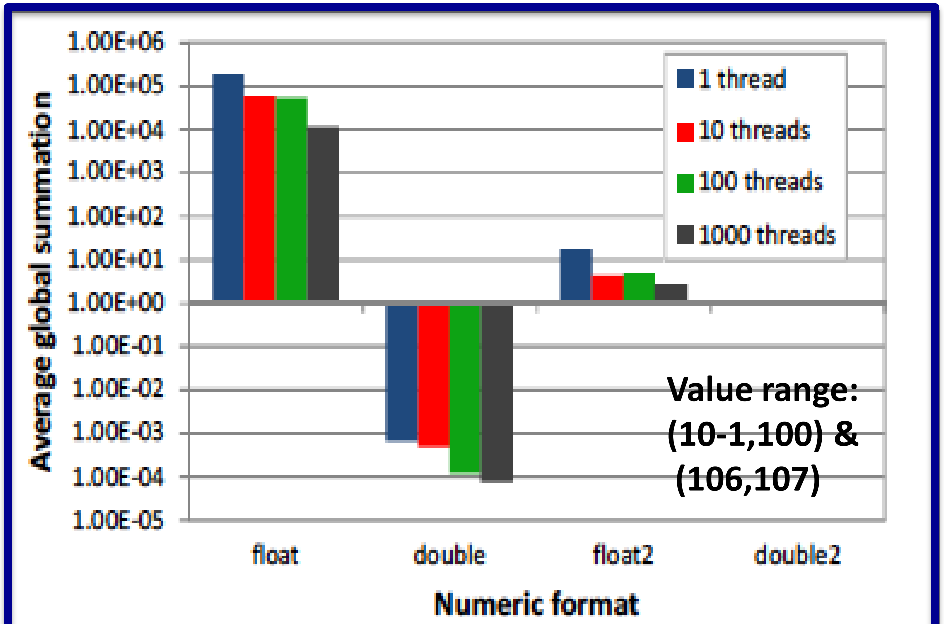


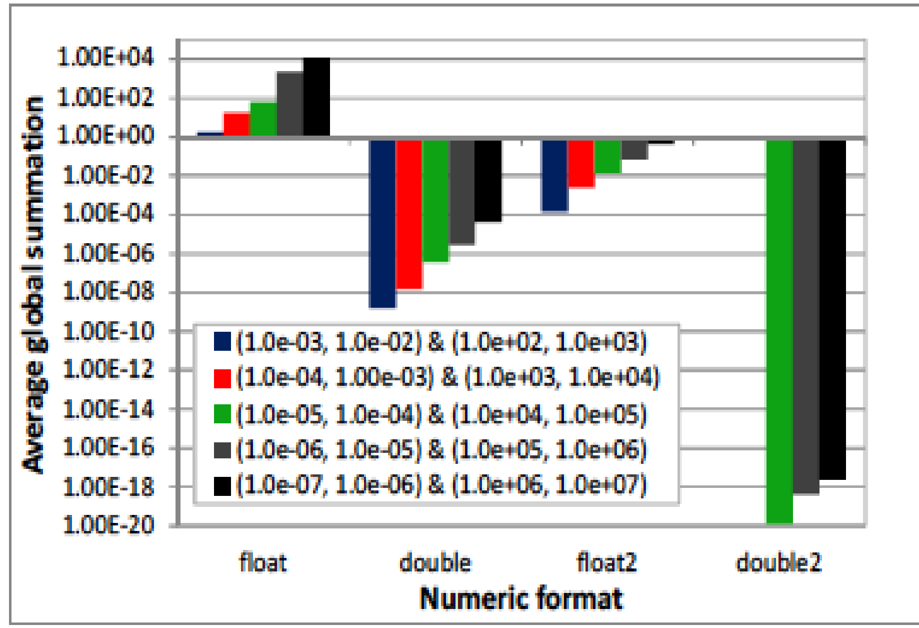


Double precision (double) shows drastic accuracy improvement
Composite precision (double2) allows fully accurate results

Accuracy on Kepler GPUs

Bars represent average absolute values of global summation over 4 runs
The expected result is 0: the smaller value, the better accuracy

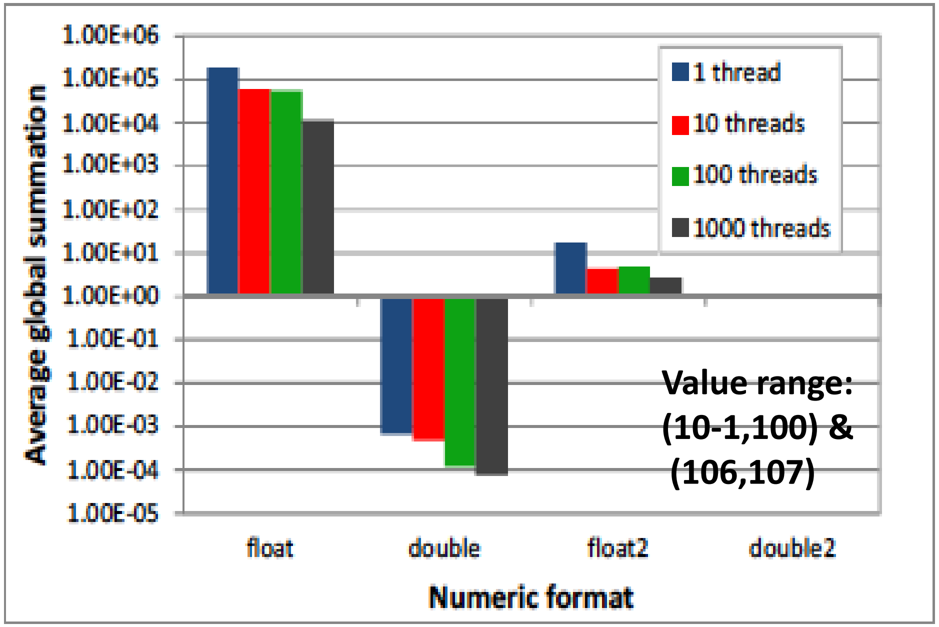


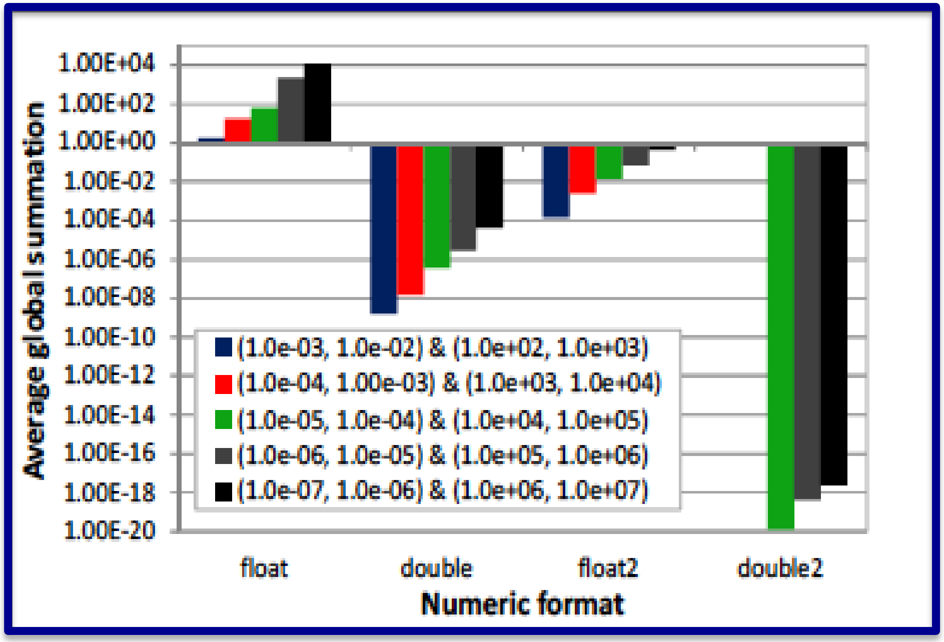


Higher multithreading degrees lead to an improvement in accuracy

Accuracy on Kepler GPUs

Bars represent average absolute values of global summation over 4 runs
The expected result is 0: the smaller value, the better accuracy

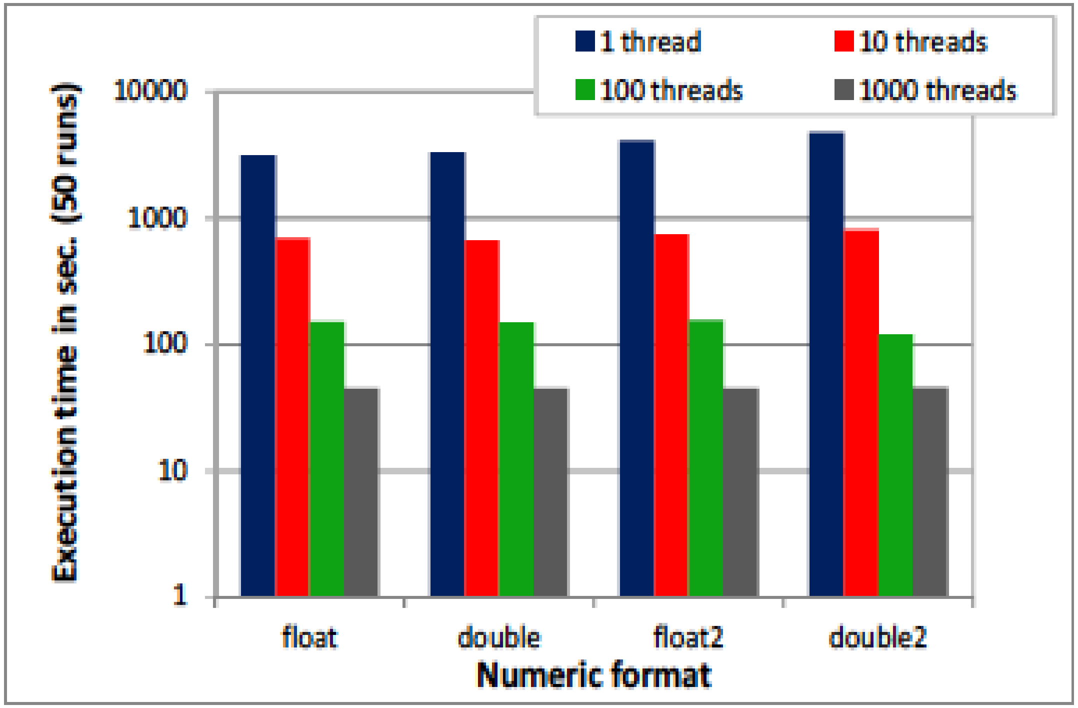




Double2 is still the preferable representation; the reported accuracy, decreases as difference in order of magnitude of input data grows

Performance on Kepler GPUs

Bars represent the average runtime in seconds of global summation over 50 runs

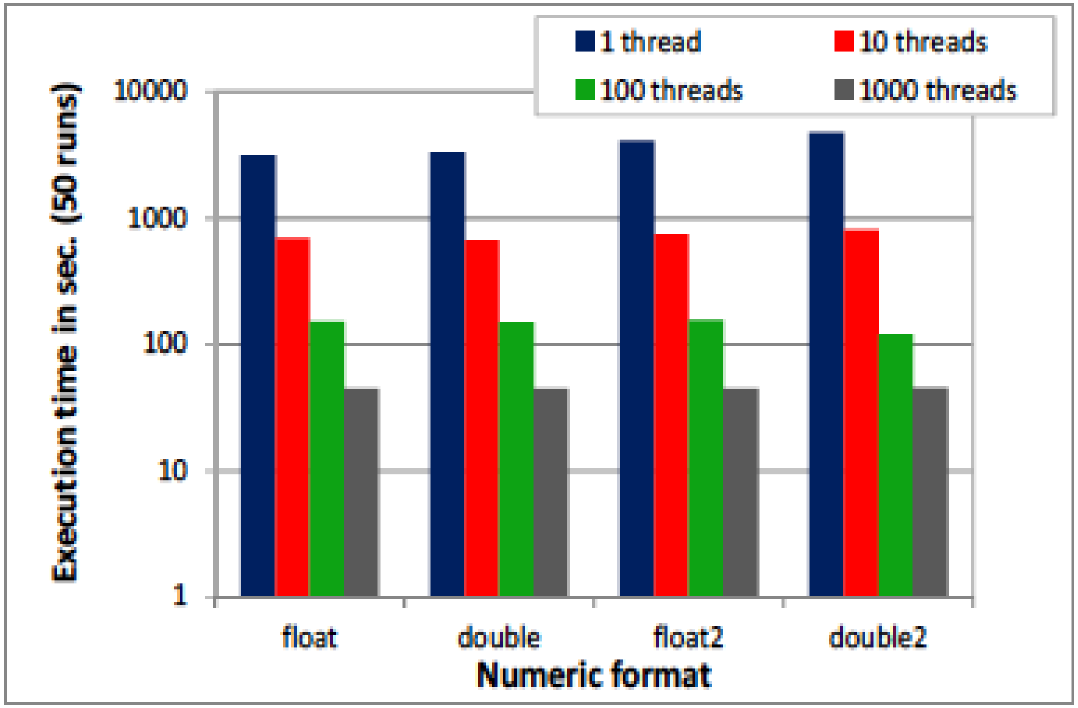


Runtime overhead of composite precision is hidden by ILP and DLP

UNIVERSITY *of* DELAWARE

Performance on Kepler GPUs

Bars represent the average runtime in seconds of global summation over 50 runs

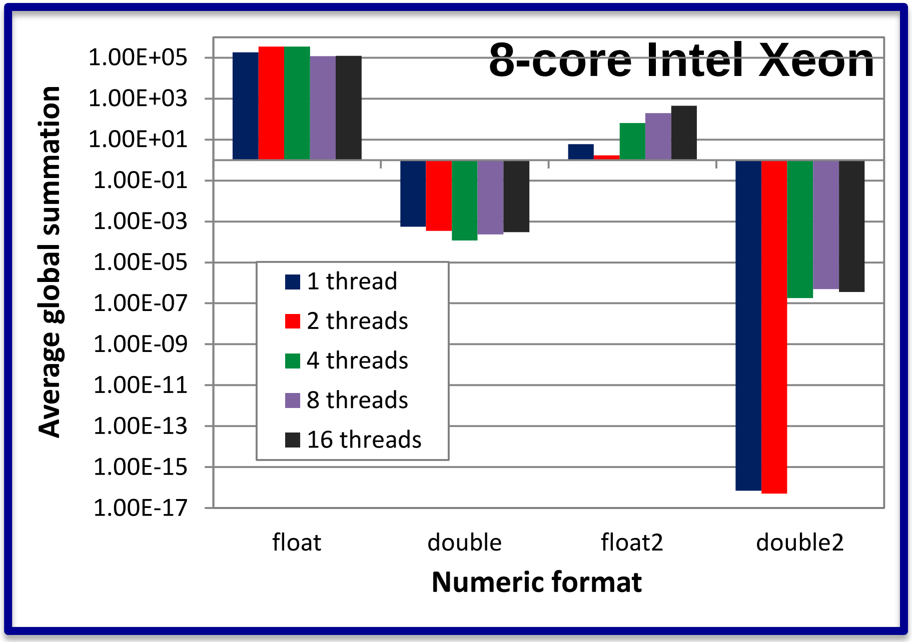


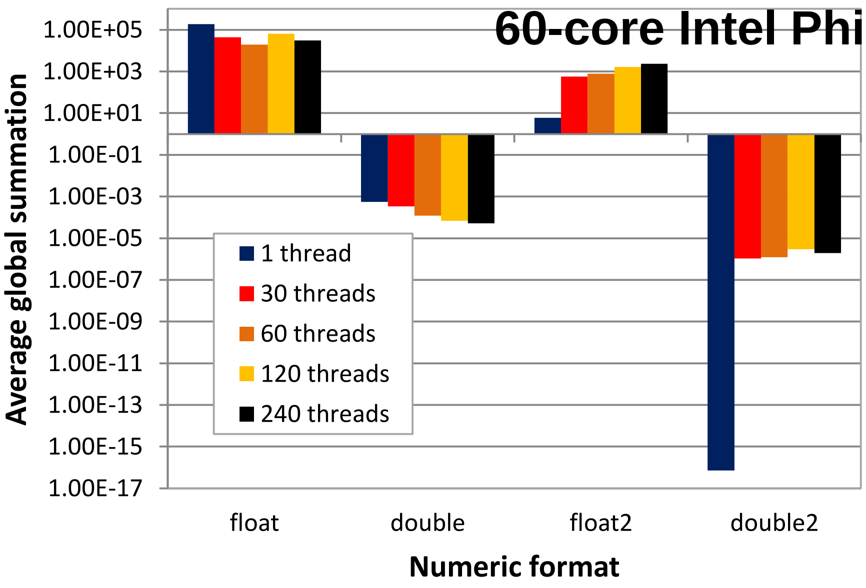
The same tests using the CUMP library exhibit 14x slow-down in case of sequential execution and 500x slow-down when running with 100 threads

Runtime overhead of composite precision is hidden by ILP and DLP

Accuracy on Multi-core CPUs and Intel Phi

Bars represent average absolute values of global summation over 4 runs
The expected result is 0: the smaller value, the better accuracy

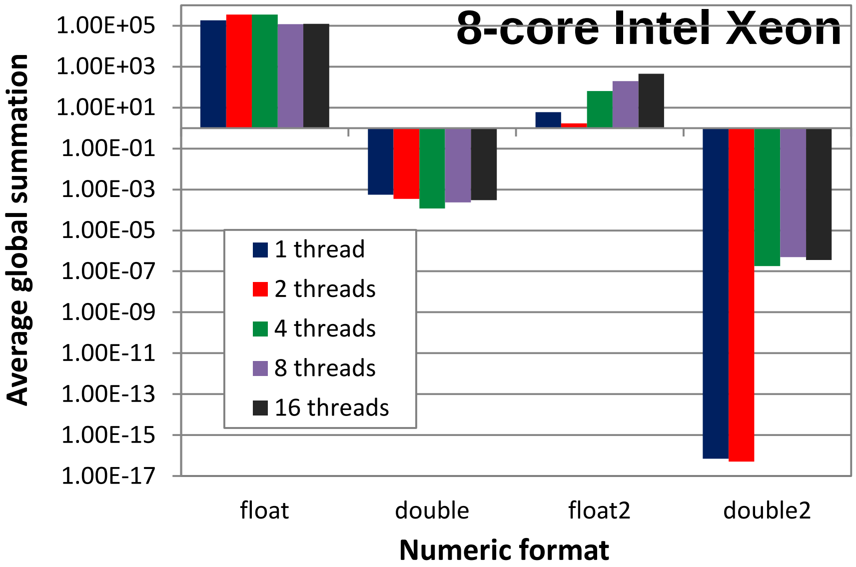


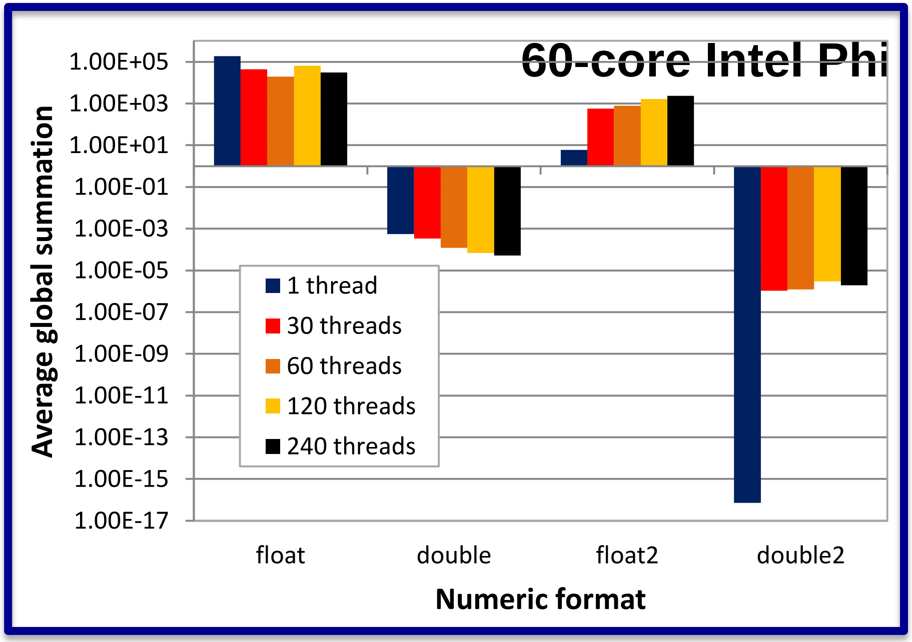


Composite precision outperforms single and double precisions but increasing multithreading makes its accuracy worse

Accuracy on Multi-core CPUs and Intel Phi

Bars represent average absolute values of global summation over 4 runs
The expected result is 0: the smaller value, the better accuracy





Composite precision outperforms single and double precisions but increasing multithreading makes its accuracy worse

Lessons Learned and Future Directions

Lessons learned:

- The size of the array, the number of threads, and the work per thread affect the precision even of sequential code
- The range of numbers affect drifting from expected result
- The performance of double precision operations have substantially improved in later GPU generations
- Intel Phi accuracy is significantly reduced by multithreading

Future directions:

- Extend the study to other techniques based on error-free transformations:
  - Kahan and Pre-Rounded Reproducible Summation
- Understand how threads-to-core mapping schemes affect accuracy on accelerators

Acknowledgments

Contacts:

taufer@acm.org
becchim@missouri.edu

Sponsors: