Discover where differences arise in scientific codes between CPU and GPU implementations and evaluate the effects on performance and accuracy.

**Myths about Debugging Scientific Code**

Many scientists do not thoroughly debug their code or compare results between GPU and CPU due to the assumption that results will differ on different platforms. Our research shows that the results helps to debug both CPU code and GPU code. By understanding the sources of differences, errors can be corrected. In the medical reconstruction code, we studied, all differences were removed and a bug was fixed in the version of the code.

---

**What is Floating-point?**

Floating-point is the scientific notation format for binary values. It allows for very large and very small numbers to be represented with one data type. A number is represented by an exponent and a mantissa, usually normalized.

- Example Equation
  - CPU: `1.001e+01 + 0.001e+01 + 0.0001e+01 + 0.00005e+01 + 0.00005e+01`
  - GPU: `1.001e+01 + 0.001e+01 + 0.0001e+01 + 0.00005e+01 + 0.00005e+01`

---

**Associativity – Accumulation vs. Reduction**

Manycore introduces new concepts of programming. For example, accumulating a large number of values on a serial CPU may involve a serial loop. Once the CPU sum gets large, it does not have enough precision to reflect the addition of a small value. The GPU bins its results by doing a reduction - consistently adding similar sized values. The GPU is faster and more accurate!

- Example Equation
  - CPU: `100.0 + 0.01 + 0.001 + 0.0001 + 0.00005 + 0.00005`
  - GPU: `100.0 + 0.01 + 0.001 + 0.0001 + 0.00005 + 0.00005`

---

**Differences Between GPU and CPU code**

Common sources of differences between GPU and CPU code:
- Reordering of instructions: Floating-point is not associative
- Implementation of instructions
  - Fused Multiply Add is implemented in NVIDIA commodity compute capability 2.0 and above, not (currently) in x86 CPUs
  - Choices made by the compiler
    - Fusing mutl-add and reordering instructions are examples

How to identify differences and debug your code:
- Compare intermediate results and identify where the divergence occurs.
- Double precision does not fix the problem!
- It only makes them less apparent, which makes them harder to recognize.
- Single precision is used for scientifically important code, even though NVIDIA assumes it is not -- scientists analyze range of values.

---

**Fixed-point to Floating-point Boundaries**

- A common source of errors is the boundary between floating and fixed point code. Floating point values are often cast to integers for image display. Errors are easily introduced. Correct casting should be used.
- Different modes of division - The default on compute capability 1.3 GPUs is approximate division in single precision. Using default conditions, the CPU is more accurate.

---

**Test Example - Tomosynthesis**

- Digital Breast Tomosynthesis (DBT) is a mammography algorithm that creates a 3D image from the data of 15 X-ray scans to aid in the search for cancerous tissues.
- Original code was developed by MGH and ported to the GPU at Northeastern by Schaan et al [4].

---

**Results of CPU and GPU Tomosynthesis**

- We match the GPU code to the CPU code in single and double precision. Each change lowers total number of differences shown in Fig. 11.
- We created GPU code that produces identical results to the CPU code.

---

**Future Work**

- Provide better debugging tools for floating point implementations on GPUs.

---

Acknowledgement: This work was supported in part by the National Science Foundation Engineering Research Centers Innovations Program (Award Number EEC-0946463).