Arithmetic for Sparse Integers and Floats

Midshipman 1/C Mark Atkins, James Browning III, and Norman Overfield
Assistant Professor Daniel S. Roche, Computer Science

Why compute with REALLY big numbers?

Needed for cryptography and numerical computing. Such applications require numbers containing thousands or even millions of bits, traditional representation is both slow and inefficient for storage.

Current software exists only for DENSE integers

Every bit must be stored in a traditional representation. Libraries such as GMP and Givaro are used for computing.

Can we do better when integers are SPARSE?

We do not have to store runs of 1's or 0's in sparse representation. Only separate strings of 1's and the 0's between each set need to be stored. This requires less space to store our numbers.

Our Code Contributions

- **Lint** - C++ large integer class. Simple implementation of a dense large integer library.
- **Splint** - C++ sparse integer class.
- **fpSplint** - C++ fixed precision floating point arithmetic library implemented using sparse integers.

What are "Sparse" Integers?

- Base 10 integer: 1112455258097
- Base 2 dense representation (41 bits): 10000001100000011100000000001111111110001
- Sparse representation
  8 bit *window size*,
  5 *nodes*.

-15	1	7	3	1
0	13	10	9	8

Window Size: The maximum number of bits in each coefficient (-15, 1, 7, 3, and 1 in this example). With window size we can store values between -128 and 127. If there is overflow, a new node will be created.

Nodes: Nodes default to store coefficients up to 32 bits, and differences up to 16 bits. Customizable through templates.

Storage: Nodes are stored in vectors, with each node containing a coefficient value, and the difference. Which is the length to the start of the next node.

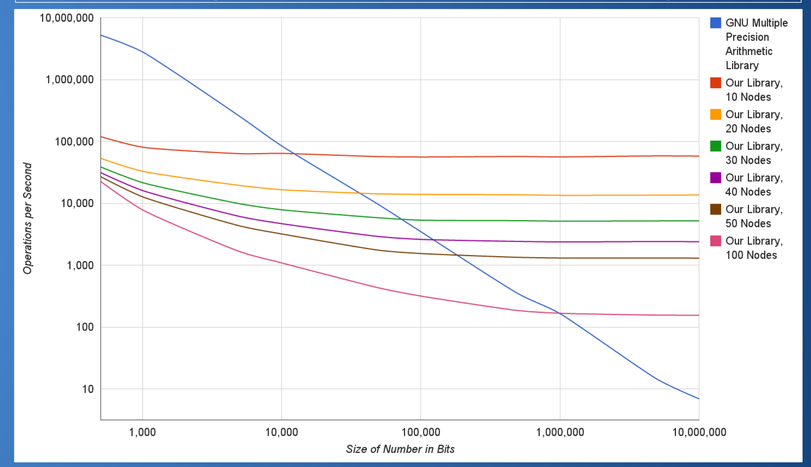
Fixed Precision Floating Point Arithmetic

The Idea: The MPFR library implements floating point calculations using GMP; we achieved this functionality using sparse integers.

Composition: We use a splint type to store our mantissa, with a long integer holding our exponent.

Capability: We are able to extend our splint functionality to make floating point calculations.

Timing comparison for GMP vs. our library



What can we do with them?

Conversion of "dense" integer: We take a dense number, and through a process involving modulo, subtraction, and right shifting, produce the optimal sparse representation.

Addition/Subtraction: Addition is similar to merging. We do this by adding up the nodes of two sparse integers one at a time until the sum is calculated. The two least significant nodes are compared, and whichever has the smallest difference value is added to the sum.

Multiplication: Calculated by multiplying one node of one sparse integer by every node in the other sparse integer. This is done for all nodes, and all of these are summed.

Conclusion

- We created three implementations for dense, sparse and floating point numbers.
- Our splint class was extensively tested against the state-of-the-art GMP library.
- The current crossover point where sparsity "wins" is around 1 million bits and 100 sparse *nodes*.
- Sparse integers and floating point numbers may have applications in improving cryptography and numerical computation.
- More work is needed to determine the feasibility of these applications.

This material is based upon work supported by the National Science Foundation under Grant No. 1319994, "AF: Small: RUI: Faster Arithmetic for Sparse Polynomials and Integers".