Survey of Optimized Functional Data Structures

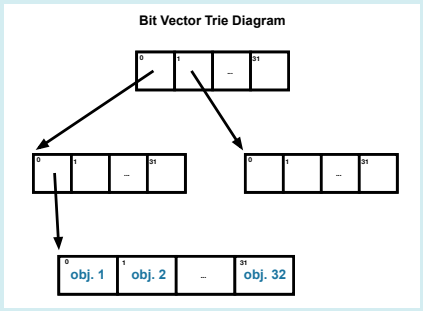
Mermer Dupree, Advisor: John Lillis

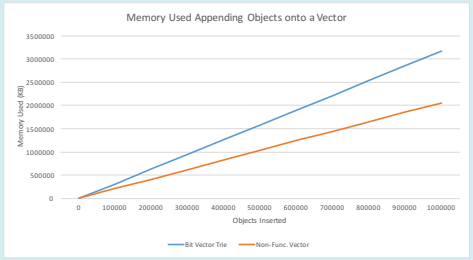
Abstract

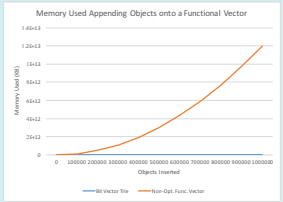
As computers become highly parallelized and memory becomes faster and cheaper, we must begin to consider new programming paradigms, like functional programming, to tackle the challenges that arise from these systems. Functional programming has for a long time been a programming paradigm of interest to computer scientists. It offers many benefits like safe multithreading, less error prone code and even the possibility of automatic parallelism. One of the challenges is that it uses high amounts of memory. Using functional data structures that are optimized can reduce the memory usage significantly. In our research, we implemented four optimized functional data structures and compared their efficiency to similar non-optimized functional data structures and non-functional data structures.

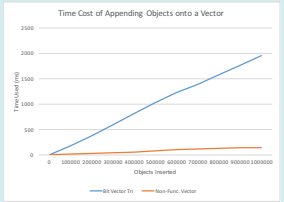
Bit Vector Trie

- Used as a Vector data structure: random access and fast append.
- A tree where the leaves are arrays of 32 objects.
- The tree has a branching factor of 32, therefore for data that could fit in memory, the height does not exceed 6.
- During updates, only arrays along the path from the root to the target location. require copying.
- This results in almost constant time performance for random updates and append.









Results

- Performed tests appending up to a million objects to bit vector trie and compared its performance to the C++ non functional vector.
- Memory performance of appends between the bit vector trie and the non-functional vector show small difference. This suggests overhead is minimal.
- Comparing the memory performance to that of a non-optimized functional vector shows drastic improvement.
- It is somewhat slower than a non-functional vector, but not by more than a constant factor.
- Other functional data structures that we tested were linked lists, banker's queues and red-black trees.

Conclusions

- Performance of bit vector trie is comparable to non-functional vector.
- Optimizations like the bit vector trie are essential for functional programming.
- Future work includes: (i) testing the bit vector trie's performance as a hash table, and (ii) testing caching's effect on speed.

UIC Department of Computer Science
UNIVERSITY OF ILLINOIS AT CHICAGO
COLLEGE OF ENGINEERING

Mermer Dupree: mdupre2@uic.edu, University of Illinois at Chicago
John Lillis: lillis@uic.edu, University of Illinois at Chicago