kmap manual

Introduction:

Kmap and kvector were designed as replacements for std::map and std::vector for handling large sized data structures. From testing, std::map and std::vector were found to be inefficient at memory management when storing large amounts of data. After further research the issue was found to be due to std::allocator which is a pool based allocator. Pool allocators store more memory than is actually needed by the application which allows memory allocation for the application to be faster. Unfortunately for large data structures, the memory bloat can fill up all of your ram. Kmap and kvector were designed to store large data structures as efficiently as possible.

Kvector also solved a key issue that std::vector does not do a very good job of solving. When std::vector is cleared, the allocated memory for the vector is not cleared. The only way to clear the memory for std::vector is to swap an empty vector with the full vector like so:

```
std::vector v1;
v1.resize(200);
{
	std::vector v2;
	v1.swap(v2);
}
```

Kvector allows the allocated memory to be cleared with one command.

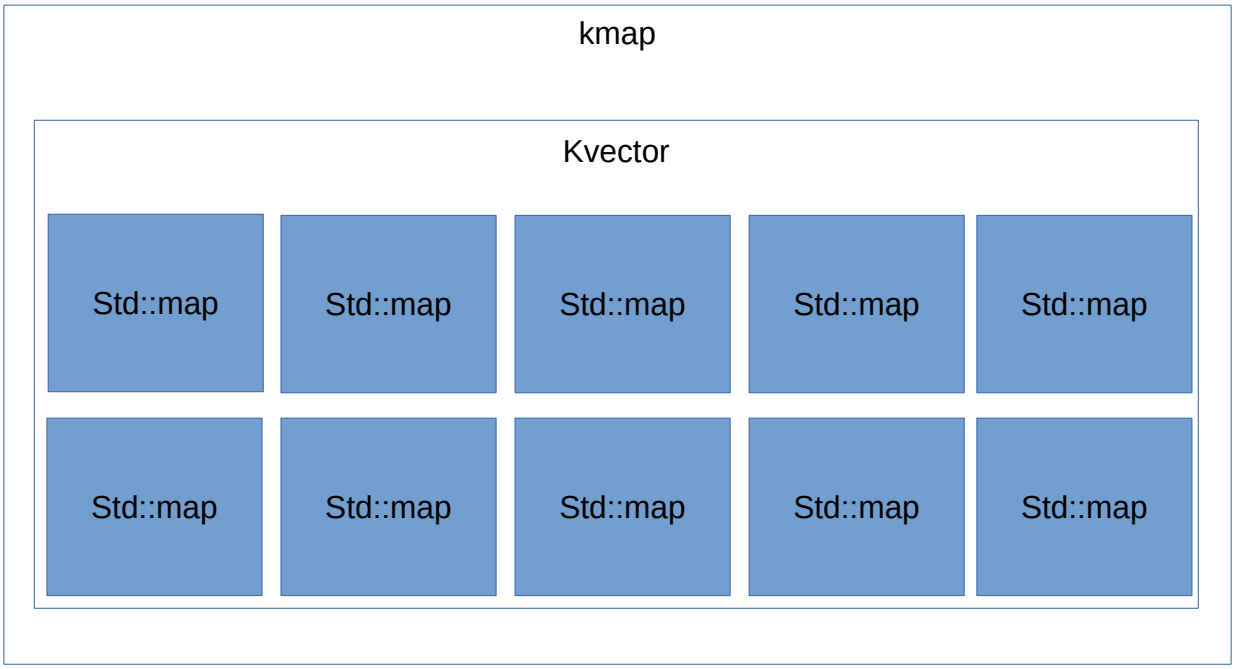

Figure 1: Diagram of the kmap data structure showing kvector storing std::map.

Kmap combines kvector with std::map (see Figure 1). Each std::map structure is deigned to contain approximately 64 keys. This creates two advantages; one, the data structure is relatively flat, and two, std::allocator works extremely efficiently for std::map. Additionally, kmap uses a hash table to insert the key-value pair in the appropriate std::map. The insert operation is assumed to be O(log(64)), since the index in kvector is assigned using a hashing function and there are at most 64 keys in each std::map. The read operation is amortized O(1) because on average Y steps are needed to find the key-value pairs where Y is independent of the N key-value pairs stored in kvector. This occurs because the key-value pairs are hashed and std::map contains on average 64 key-value pairs under a well distributed hashing.

Kmap uses an iterator called kmap_iterator. Kmap_iterator stores the index of kvector and an iterator pointing to a key-value pair stored in std::map. Unlike iterators for std::map and std::vector, this iterator is not designed to be used with methods contained in the algorithms header, part of the c++ standard library.

kmap_iterator Methods:

❖bool operator != (uint64_t): tests if kmap_iterator has reached the end of kvector.

kmap Methods:

❖kmap(): constructor that creates the default sized kmap
❖kmap(uint64_t): constructor that creates a specifically sized kmap
❖kmap(const kmap<K,V>&): copy constructor
❖kmap<K,V>& operator=(const kmap<K,V>&): assignment operator
❖void clear(): replaces old kmap with an empty default sized kmap
❖void clean(): clears all key-value pairs from kmap
❖void insert(const K&, const V&): inserts key-value pair into kmap
❖ V& operator[](const K&): if key exists inserts value into kmap otherwise returns the value assigned to the key
❖void resize(uint64_t): sizes kmap to handle the number in the resize command. If the number of entries the current kmap can handle is greater than the value given in the command nothing happens. Otherwise kmap is resized and rehashed.
❖void swap (kmap &): swap between two kmaps
❖kmap_iterator<K,V> findkey(const K&): returns a kmap_iterator containing the hash value of the key and a std::map iterator pointing to the key in std::map.
❖void removekey(const K&): removes the key from kmap. If the key does not exist, no changes will occur to the kmap.
❖const K& getkey(const kmap_iterator<K,V> &): gets the key from kmap corresponding to the current value of kmap_iterator. This will throw an error if the iterator is at the end of std::map. This method is designed to be combined with kmap's iterating methods or findkey.
❖ V& getvalue(const kmap_iterator<K,V> &): gets the value from kmap corresponding to the current value of kmap_iterator. This will throw an error if the iterator is at the end of std::map. This method is designed to be combined with kmap's iterating methods or findkey.
❖void setvalue(const kmap_iterator<K,V> &, const V&): sets the value from kmap corresponding to the current value kmap_iterator. This will throw an error if the iterator

is at the end of std::map. This method is designed to be combined with kmap's iterating methods or findkey.
- ❖kmap_iterator<K,V> start(): Finds the first std::map in kvector that is not empty and sets kmap_iterator to the beginning of std::map. Kmap_iterator is also assigned the index of the first non-empty std::map. If none of the std::maps are filled, this method sets the index in kmap_iterator to the value returned by the method end.
- ❖uint64_t end(): returns the position after the end of kvector in kmap. The returned value is also the capacity of the kvector in kmap.
- ❖void next(kmap_iterator<K,V> &): Sets kmap_iterator to the next value in std::map. If std::map's next value is the end, goto the next filled std::map and set kmap_iterator to the beginning of the std::map.
- ❖bool empty(): returns true if kmap is empty. Otherwise, return false.

kvector Methods:
- ❖kvector(): default constructor which sets kvector to the default size
- ❖kvector(uint64_t): constructor which sets the size of kvector to the specified size
- ❖kvector(uint64_t, T): constructor which sets the size of kvector the specified size and fills the kvector with the specified value.
- ❖kvector(const kvector &): copy constructor
- ❖void resize(uint64_t): resize kvector to be the specified size. Note: this can shrink the kvector which can cause loss of information but will not cause a memory leak.
- ❖void clear(): deletes kvector and creates a new kvector with size and capacity of 0.
- ❖T & operator [] (uint64_t): returns the value stored at the kvector's index.
- ❖const T & operator [] (uint64_t) const: returns an unmodifiable value stored at the kvector's index.
- ❖void push_back(T): add a value to the end of the kvector and increase the size of the kvector by one.
- ❖void pop_back(): removes the value at the end of the kvector and decrease the size of the kvector by one.
- ❖uint64_t getsize() const: returns the size of kvector.
- ❖uint64_t getcapacity() const: returns the capacity of kvector.
- ❖bool empty() const: returns true if the kvector is empty. Otherwise, this method returns false.
- ❖void clean(): deletes the previous kvector and creates a new kvector with the same capacity as the old kvector.
- ❖void swap(kvector<T> &): swap between two kvectors
- ❖kvector<T> & operator= (const kvector<T> &): assignment operator
- ❖T & back(): returns the value at the end of kvector.
- ❖const T & back() const: returns an unmodifiable value at the end of kvector.
- ❖T & front(): returns a value at the front of kvector.
- ❖const T & front() const: returns an unmodifiable value at the front of kvector.