

---

Subject: InVector committed

Posted by [mirek](#) on Sat, 16 Feb 2013 16:50:05 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

U++ just got 5 new containers, all based on a new idea about how to implement fast-insertion vector:

InVector - fast insertion vector

InArray - ..and its Array flavor

SortedIndex - index that keeps keys in sorted order and uses binary search

SortedVectorMap - ...its (In)Vector Map derivative

SortedArrayMap - ...(In)Array derivative

Interface-wise, InVector behaves just like normal Vector (give or take some methods). Insertion times are fast, as demonstrated by benchmark that takes array of N elements, then inserts 10000 elements at position 0, one by one, then removes all of them in single go (and that done 100 times to get some meaningful numbers) (all tests performed with 64bit linux):

```
[In]Vector<String> o;
for(int i = 0; i < n; i++)
    o.Add(AsString(i));
String h = "0";
TimeStop tm;
for(int i = 0; i < 100; i++) {
    for(int j = 0; j < 10000; j++)
        o.Insert(0, h);
    o.Remove(0, 10000);
}
```

-----  
InVector:

1000: 136 ms

2000: 135 ms

5000: 138 ms

10000: 143 ms

20000: 151 ms

50000: 165 ms

100000: 188 ms

200000: 218 ms

500000: 276 ms

1000000: 333 ms

2000000: 403 ms

5000000: 510 ms

-----  
Vector:

1000: 3839 ms  
2000: 4560 ms  
5000: 6702 ms  
10000: 10377 ms  
20000: 19982 ms  
50000: 48384 ms

(after 50000, it got painfully slow for Vector).

Worst case for index retrieval (e.g. `operator[]`) is  $\log(n)$ , but it is quite fast in real situation. For simple linear scans over single `InVector`, you can expect `operator[]` to be 3 times slower than on for `Vector::operator[]` (thanks to per-thread caching). In the very worst case, it can be about 30 times slower for any realistic element counts (tens of millions). But keep in mind that `Vector::operator[]` is extremely fast... Iterators to `InVector` are similar, linear scans are very fast again.

`InVector` has optimized `Find[Upper/Lower]Bound` methods which return index and have  $\log(n)$  worst case.

Moving on to associative variants, they are about as fast as node based binary trees (`std::set`, `std::map`) for any realistic element counts (again, tens of millions). Inserts are bit (~30%) slower for very large element counts (millions), searches are quite (~50%) faster. Benchmark numbers for various data types for benchmark that scans 3MB for frequency of all words (23000 unique words in book total) (scan repeated 10 times to get meaningful numbers):

`std::map<std::string, int>` time: 3131 ms  
`std::map<String, int>` time: 2041 ms  
`SortedVectorMap<String, int>` time: 1005 ms  
`SortedArrayMap<String, int>` time: 1045 ms  
`VectorMap<String, int>` time: 378 ms

Header for new containers is in `Core/InVector.h`.

Tests:

`upptst/InVector`  
`upptst/InArray`  
`upptst/SortedIndex`  
`upptst/SortedAMap`

Benchmarks:

`benchmarks/InVector`  
`benchmarks/InVectorIR`  
`benchmarks/AllMaps`

Maturity status: Despite rigorous testing I would not hurry to replace all Vectors with InVectors where ever benefit can be expected, but I would dare to use them in the new yet untested code.

TODO: Docs...

---

---

Subject: Re: InVector committed  
Posted by [Lance](#) on Sun, 03 Mar 2013 13:43:05 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hi Mirek:

Please briefly explain the tradeoff? I mean, comparing to Vector, does InVector pay some other prices, eg, more memory footprint, for dramatically increased insertion performance?

What's the situations that we should favor one over another?

Thanks,

Lance

---

---

Subject: Re: InVector committed  
Posted by [mirek](#) on Sun, 03 Mar 2013 16:41:42 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Lance wrote on Sun, 03 March 2013 08:43Hi Mirek:

Please briefly explain the tradeoff? I mean, comparing to Vector, does InVector pay some other prices, eg, more memory footprint, for dramatically increased insertion performance?

Memory footprint is not dramatically different. What is slower is element retrieval.

Quote:

What's the situations that we should favor one over another?

Obviously, if you are using Insert/Remove at arbitrary position and the Vector has more than 2KB of data, InVector will have faster inserts.

If you are not using Insert (which IMO is still majority of cases), use Vector.

If you need map with range search, you obviously have to use Ordered index/maps.

Mirek

---

---

Subject: Re: InVector committed  
Posted by [Lance](#) on Mon, 04 Mar 2013 14:23:50 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Thanks, Mirek.

---

---

Subject: Re: InVector committed  
Posted by [busiek](#) on Tue, 02 Apr 2013 12:24:02 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

mirek wrote on Sat, 16 February 2013 17:50U++ just got 5 new containers, all based on a new idea about how to implement fast-insertion vector

Interesting. I wonder what is the new idea? Can you give some reference or sketch? I tried to read the code, but some hints would be useful.

---

---

Subject: Re: InVector committed  
Posted by [dolik.rce](#) on Tue, 02 Apr 2013 13:21:49 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

busiek wrote on Tue, 02 April 2013 14:24mirek wrote on Sat, 16 February 2013 17:50U++ just got 5 new containers, all based on a new idea about how to implement fast-insertion vector

Interesting. I wonder what is the new idea? Can you give some reference or sketch? I tried to read the code, but some hints would be useful.  
You can find some more details in this thread.

Honza

---