Subject: A new container in works: Flex - fast insertion vector
Posted by mirek on Wed, 29 Aug 2007 17:00:33 GMT
View Forum Message <> Reply to Message

Recently I have enjoyed some time implementing my former idea of novelty random access container (aka indexed) container, similar to Vector or std::vector, which has much improved time of insertion of element at arbitrary position, while keeping operator[] complexity at O(1).

It was a success, my idea seems to work as expected:

I have benchmarked it by creating a Vector/std::vector/Flex by inserting "size" elements at random positions:

int

size (times)	std::vector	Vector	Flex
10 (100000x)	0.078 s	0.016 s	0.031 s
20 (50000x)	0.078 s	0.032 s	0.047 s
40 (25000x)	0.062 s	0.047 s	0.047 s
80 (12500x)	0.078 s	0.047 s	0.062 s
160 (6250x)	0.094 s	0.047 s	0.078 s
320 (3125x)	0.109 s	0.063 s	0.094 s
640 (1562x)	0.140 s	0.110 s	0.125 s
1280 (781x)	0.203 s	0.156 s	0.172 s
2560 (390x)	0.312 s	0.282 s	0.203 s
5120 (195x)	0.562 s	0.516 s	0.234 s
10240 (97x)	1.078 s	1.032 s	0.250 s
20480 (48x)	2.547 s	2.500 s	0.312 s
40960 (24x)	5.625 s	5.594 s	0.437 s
81920 (12x)	11.672 s	11.625 s	0.578 s
163840 (6x)	23.703 s	23.657 s	0.703 s
327680 (3x)	47.688 s	47.656 s	0.953 s

String

size (times)	std::vector	Vector	Flex
10 (100000x)	0.125 s	0.063 s	0.094 s
20 (50000x)	0.140 s	0.078 s	0.078 s
40 (25000x)	0.141 s	0.063 s	0.078 s
80 (12500x)	0.157 s	0.093 s	0.110 s
160 (6250x)	0.218 s	0.125 s	0.125 s
320 (3125x)	0.329 s	0.187 s	0.188 s
640 (1562x)	0.562 s	0.313 s	0.265 s
1280 (781x)	1.063 s	0.593 s	0.297 s
2560 (390x)	2.047 s	1.203 s	0.344 s
5120 (195x)	3.938 s	2.796 s	0.422 s
10240 (97x)	7.797 s	6.235 s	0.500 s

20480 (48x)	15.140 s	12.641 s	0.656 s
40960 (24x)	30.234 s	25.688 s	0.969 s
81920 (12x)	60.703 s	51.765 s	1.313 s
163840 (6x)	149.062 s	135.922 s	2.078 s
327680 (3x)	433.078 s	411.735 s	3.734 s

Means it is as almost as fast as Vector to about 1000 elements, then it starts to outperform it more and more...

There is a lot of work left before this can be introduced to the Core (it is hard as hell to implement, my productivity doing this is about 10 lines / day, so far I have only implemented Insert, Remove will cost me another day or two... , but nevertheless I am pleased...

Mirek

Subject: Re: A new container in works: Flex - fast insertion vector
Posted by unodgs on Wed, 29 Aug 2007 19:13:05 GMT

View Forum Message <> Reply to Message

Excelent results! Should I use it in GridCtrl instead of Vector< Vector<Value> > to store grid data? Are there any cons? With Vector insertion and removing row when grid has more than 500.000 rows is visibly slow.

Subject: Re: A new container in works: Flex - fast insertion vector
Posted by mirek on Wed, 29 Aug 2007 20:22:41 GMT

View Forum Message <> Reply to Message

Well, the only drawback I am aware of is that it is this fast for adding/removing single element only.

For more elements (e.g. more lines), speed gradually drops to that of regular Vector or perhaps slightly worse (so far, this is only projected behaviour, implementing insertion of more elements will be another several days job .

Anyway, this means that you should not get worse results for these cases than with Vector.

BTW, even memory consumption is very favorable, only a percent or two more than regular vector.

Mirek

Subject: Re: A new container in works: Flex - fast insertion vector
Posted by nasos_i on Thu, 20 Sep 2007 21:02:23 GMT
View Forum Message <> Reply to Message

Why not build a wrapper on something like:
template T std::map<int, T>

?

Subject: Re: A new container in works: Flex - fast insertion vector
Posted by mirek on Thu, 20 Sep 2007 21:19:52 GMT
View Forum Message <> Reply to Message

nasos_i wrote on Thu, 20 September 2007 17:02Why not build a wrapper on something like:
template T std::map<int, T>

?

Because this has O(1) index access

("Fast" vector wrappers over map have O(log n) AFAIK).

Mirek

Subject: Re: A new container in works: Flex - fast insertion vector
Posted by sergei on Thu, 20 Sep 2007 22:44:31 GMT
View Forum Message <> Reply to Message

Have you benched this against Array? Array should be better than Vector in insert/delete (and maybe other things) for large-sized elements (not int, something like large classes).

Subject: Re: A new container in works: Flex - fast insertion vector
Posted by mirek on Thu, 20 Sep 2007 23:34:13 GMT
View Forum Message <> Reply to Message

sergei wrote on Thu, 20 September 2007 18:44Have you benched this against Array? Array should be better than Vector in insert/delete (and maybe other things) for large-sized elements (not int, something like large classes).

Sergei, I have designed Array, I am well aware about that

But you can cheat the O numbers only to some degree. Both Array and Vector have basic insertion complexity O(n), while this new beast has something like O(sqrt(N)).

Note also that once you have base Flex implemented, adding ArrayFlex is trivial..

Mirek

Subject: Re: A new container in works: Flex - fast insertion vector
Posted by sergei on Thu, 20 Sep 2007 23:52:40 GMT
View Forum Message <> Reply to Message

luzr wrote on Fri, 21 September 2007 01:34sergei wrote on Thu, 20 September 2007 18:44Have you benched this against Array? Array should be better than Vector in insert/delete (and maybe other things) for large-sized elements (not int, something like large classes).

Sergei, I have designed Array, I am well aware about that

But you can cheat the O numbers only to some degree. Both Array and Vector have basic insertion complexity O(n), while this new beast has something like O(sqrt(N)).

Note also that once you have base Flex implemented, adding ArrayFlex is trivial..

Mirek

O(sqrt(N))? Nice... Are there downsides vs. Vector for this container (besides slightly slower on small-sized containers)? And does it basically require moveable?

P.S. Are there any sorted containers in U++, like set/multiset/map/multimap? I prefer that over GetSortOrder.

Subject: Re: A new container in works: Flex - fast insertion vector
Posted by mirek on Fri, 21 Sep 2007 07:00:55 GMT
View Forum Message <> Reply to Message

sergei wrote on Thu, 20 September 2007 19:52luzr wrote on Fri, 21 September 2007 01:34sergei wrote on Thu, 20 September 2007 18:44Have you benched this against Array? Array should be better than Vector in insert/delete (and maybe other things) for large-sized elements (not int, something like large classes).

Sergei, I have designed Array, I am well aware about that

But you can cheat the O numbers only to some degree. Both Array and Vector have basic insertion complexity O(n), while this new beast has something like O(sqrt(N)).

Note also that once you have base Flex implemented, adding ArrayFlex is trivial..

Mirek

O(sqrt(N))? Nice... Are there downsides vs. Vector for this container (besides slightly slower on small-sized containers)?

Sure While operator[] is O(1), it is slower than both Vector::operator[] and Array::opertor[] (BTW, Array::[] is much slower than Vector::[] too).

Also, appending at the end can be slower too.

Quote:
And does it basically require moveable?

I guess that "vector-flavor" will. In fact, I am considering to introduce up to 4 variants with varying characteristics.

Quote:
P.S. Are there any sorted containers in U++, like set/multiset/map/multimap? I prefer that over GetSortOrder.

No. But that is definitely one of potential uses of this new container. I have already tried to implement sorted array with this and benchmarked it against set, for 160 000 String elements, "unique" operation is about 2.5x slower than with std::set, while find only scenario is even a bit faster. Note also that std::set was using U++ allocator in this test; with vendor allocators it would be slower.

Mirek

Subject: Re: A new container in works: Flex - fast insertion vector
Posted by sergei on Fri, 21 Sep 2007 11:51:40 GMT
View Forum Message <> Reply to Message

When I said sorted containers what I needed is not find/unique (hash could do that too), but to iterate over all elements, from smallest ro largest / reverse. You say operator[] is slow. Would such iteration be slower than in set, or not?

I don't know how allocators work, but what have you done to get it faster than default? Some kind of buffering to allocate once for several small elements?

Subject: Re: A new container in works: Flex - fast insertion vector
Posted by mirek on Fri, 21 Sep 2007 12:30:16 GMT
View Forum Message <> Reply to Message

sergei wrote on Fri, 21 September 2007 07:51When I said sorted containers what I needed is not find/unique (hash could do that too), but to iterate over all elements, from smallest ro largest / reverse. You say operator[] is slow. Would such iteration be slower than in set, or not?

Not actually measured, but iterating through link structures tends to be slow with modern CPUs, so I would say that the linear iteration would be about the same.

That said, it is a bit surprising that you would prefer sorted container only to get items in order. Using Index (or generally a hash_map) and Sorting the result beats this in most cases.

For me, the main advantage of such container would be the lower and upper bounds.

Quote:
I don't know how allocators work, but what have you done to get it faster than default? Some kind of buffering to allocate once for several small elements?

The algorithm of latest U++ allocator is described in the Core/srcimp.tpp.

Mirek

Subject: Re: A new container in works: Flex - fast insertion vector
Posted by sergei on Fri, 21 Sep 2007 12:51:17 GMT
View Forum Message <> Reply to Message

luzr wrote on Fri, 21 September 2007 14:30sergei wrote on Fri, 21 September 2007 07:51When I said sorted containers what I needed is not find/unique (hash could do that too), but to iterate over all elements, from smallest ro largest / reverse. You say operator[] is slow. Would such iteration be slower than in set, or not?

Not actually measured, but iterating through link structures tends to be slow with modern CPUs, so I would say that the linear iteration would be about the same.

That said, it is a bit surprising that you would prefer sorted container only to get items in order. Using Index (or generally a hash_map) and Sorting the result beats this in most cases.

For me, the main advantage of such container would be the lower and upper bounds.

Quote:
I don't know how allocators work, but what have you done to get it faster than default? Some kind

of buffering to allocate once for several small elements?

The algorithm of latest U++ allocator is described in the Core/srcimp.tpp.

Mirek

Sorting would beat it for a static set. But would it, if the set was changing (adding/removing elements), and at any given time I might need the order? Add/remove is O(logn), iteration should be linear, and sorting would cause the iteration to become O(nlogn) (sort+iterate), unless add/remove maintains sort order. And even in the best case of flex, add/remove would be O(sqrtn), worse than O(logn), right?

Subject: Re: A new container in works: Flex - fast insertion vector
Posted by mirek on Fri, 21 Sep 2007 13:06:53 GMT

View Forum Message <> Reply to Message

Quote:
Sorting would beat it for a static set. But would it, if the set was changing (adding/removing elements), and at any given time I might need the order? Add/remove is O(logn), iteration should be linear, and sorting would cause the iteration to become O(nlogn) (sort+iterate), unless add/remove maintains sort order. And even in the best case of flex, add/remove would be O(sqrtn), worse than O(logn), right?

That is correct,

- there is a bit more about O - something like "generic speed". Flex seems to be faster up to 160 elements (in fact, even sorted U++ Vector is faster there than std::set).

- I believe I do a lot of programming, yet I never had encountered a problem that would require continual sorted iteration.

(BTW, I have another idea of container, something much more traditional, basically sorted variant of Index based on tree implemention; with method GetNextSorted that returns the index of next element in sequence. Given my experience with Index v.s. node based hashmap, I dare to say that such linearized tree implementation will be faster than node based.)

Subject: Re: A new container in works: Flex - fast insertion vector
Posted by sergei on Fri, 21 Sep 2007 13:17:48 GMT

View Forum Message <> Reply to Message

luzr wrote on Fri, 21 September 2007 15:06Quote:
Sorting would beat it for a static set. But would it, if the set was changing (adding/removing elements), and at any given time I might need the order? Add/remove is O(logn), iteration should be linear, and sorting would cause the iteration to become O(nlogn) (sort+iterate), unless add/remove maintains sort order. And even in the best case of flex, add/remove would be O(sqrtn), worse than O(logn), right?

That is correct,

- there is a bit more about O - something like "generic speed". Flex seems to be faster up to 160 elements (in fact, even sorted U++ Vector is faster there than std::set).

- I believe I do a lot of programming, yet I never had encountered a problem that would require continual sorted iteration.

(BTW, I have another idea of container, something much more traditional, basically sorted variant of Index based on tree implemention; with method GetNextSorted that returns the index of next element in sequence. Given my experience with Index v.s. node based hashmap, I dare to say that such linearized tree implementation will be faster than node based.)

Sorted Vector - that's sort and add sorted, or sort on every change?

As for a problem, out of my head: graph with set of elements in each node, elements move between nodes during the algorithm, output result (with sorted nodes) on each step of algorithm. There probably are other cases too.

Index with linearized tree is like heap? That would be pretty good (Index should have good "unique" time). Having a map flavor of such a container would be even better.

Subject: Re: A new container in works: Flex - fast insertion vector
Posted by mirek on Fri, 21 Sep 2007 14:36:43 GMT
View Forum Message <> Reply to Message

sergei wrote on Fri, 21 September 2007 09:17
Sorted Vector - that's sort and add sorted, or sort on every change?

That is insert at sorted position (found using binary search).

Mirek