Subject: Large data ahead
Posted by Tom1 on Tue, 29 Jan 2019 14:56:56 GMT
View Forum Message <> Reply to Message

Hi,

It seems there is 'Large data ahead' as I have already faced data sets with over 1e9 items. While the containers (Vector, Array, etc.) have an "int GetCount()", there will soon be a question about handling those large data sets as int will wrap. Obviously, these large data sets run only on 64-bit platforms and with 32+ GB of memory, but this seems to be reality already.

Is there a plan to make high item count variants of U++ containers? Or just change the current containers to be high item count compatible using int64 instead of int?

Best regards,

Tom

Subject: Re: Large data ahead
Posted by mirek on Tue, 29 Jan 2019 19:47:09 GMT
View Forum Message <> Reply to Message

Well, I acknowledge that this is an issue that needs to be considered....

The original reason for 'int' is memory consumption. In the moment that it would be changed to size_t means that all offsets everywhere are now twice as long...

For that reason I believe having "HugeVector" etc... is not that bad idea... But I might be wrong.

Mirek

Subject: Re: Large data ahead
Posted by Tom1 on Wed, 30 Jan 2019 09:22:22 GMT
View Forum Message <> Reply to Message

Hi,

How about making the containers internally int64 and then offering both int and int64 interfaces in parallel? I'm sure there would be some tight spots (with e.g. serialization to maintain binary compatibility with int indexed serialized content in a way that they maintain the old binary serialization format until the element count goes beyond reach of int.) In most cases we know for sure already that the item count will absolutely remain below 2G, so there is no point in using int64 indexing in those cases, and therefore int offsets can be retained throughout the rest of such code.

Anyway, I have no hurry for this at the moment, so let's just give it some time...

Best regards,

Tom

Subject: Re: Large data ahead
Posted by mirek on Wed, 30 Jan 2019 09:43:31 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Wed, 30 January 2019 10:22Hi,
How about making the containers internally int64 and then offering both int and int64 interfaces in parallel?

The reason I am hesitant about this is following:

```
struct Item {
  Vector<String> foo;
  Vector<int> bar;
};
```

Now with 'int' size, this is 32 bytes. With 'int64', this grows by 16 bytes (50%) which would be wasted to store zeroes in 99.999% of cases...

Also, Vector having exactly 16 bytes has (very) subtle advantage, it is a 'nice' number (memory is allocated in multiplies of 16, adressing can be done with simple shift etc...

Mirek

Subject: Re: Large data ahead
Posted by Tom1 on Wed, 30 Jan 2019 10:50:03 GMT
View Forum Message <> Reply to Message

OK, I see. I agree it is a stupid thing to waste RAM for storing a whole lot of zeros. Efficiency and speed must be considered in every step along the way and this is one of those steps.

As you pointed out, this is really a "99.999% of 32 bit and 0.001 % of 64 bit" -type of situation, so maybe I should first consider using Buffer<T> instead of Vector<T>. This may be beneficial from multiple points of view after all...

Thanks for your insight.

BR, Tom

Subject: Re: Large data ahead
Posted by mirek on Wed, 30 Jan 2019 11:09:55 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Wed, 30 January 2019 11:50OK, I see. I agree it is a stupid thing to waste RAM for storing a whole lot of zeros. Efficiency and speed must be considered in every step along the way and this is one of those steps.

As you pointed out, this is really a "99.999% of 32 bit and 0.001 % of 64 bit" -type of situation, so maybe I should first consider using Buffer<T> instead of Vector<T>. This may be beneficial from multiple points of view after all...

Thanks for your insight.

BR, Tom

Well, all that said, I am 100% OK with HugeVector (and then most of other containers) with int64 GetCount()...

Subject: Re: Large data ahead
Posted by Tom1 on Wed, 30 Jan 2019 14:07:29 GMT
View Forum Message <> Reply to Message

mirek wrote on Wed, 30 January 2019 13:09Tom1 wrote on Wed, 30 January 2019 11:50OK, I see. I agree it is a stupid thing to waste RAM for storing a whole lot of zeros. Efficiency and speed must be considered in every step along the way and this is one of those steps.

As you pointed out, this is really a "99.999% of 32 bit and 0.001 % of 64 bit" -type of situation, so maybe I should first consider using Buffer<T> instead of Vector<T>. This may be beneficial from multiple points of view after all...

Thanks for your insight.

BR, Tom

Well, all that said, I am 100% OK with HugeVector (and then most of other containers) with int64 GetCount()...

OK, Thanks Mirek. I will keep this in mind. The other points of view above include the fact that when loading huge data sets to RAM (near physical memory limits), re-allocating the buffer when Vector grows is a problem. Therefore, an accurately pre-calculated memory block size and a single allocation of that block really pays off -- both in processing time and in memory efficiency.

Then Buffer<T> is just about the right choice.

Thanks and best regards,

Tom
