Subject: Materials for articles: "U++ Core comparison to BOOST"
Posted by fudadmin on Sun, 30 Jul 2006 23:07:39 GMT
View Forum Message <> Reply to Message

from wikipedia:
Quote:
Boost provides extension libraries in the following areas:

Algorithms
Concurrent programming (threads)
Containers
Correctness and testing
Data structures
Function objects and higher-order programming
Generic programming
Graphs //aris "no in U++" remove completely?
Input/output
Interlanguage support (for Python) //aris: not yet in U++
Iterators
Math and Numerics
Memory
Misc
Parsers
Preprocessor Metaprogramming
Smart pointers (shared_ptr), with automatic reference counting[2]
String and text processing
Template metaprogramming
Workarounds for broken compilers

I think we could provide a similar list but maybe just re-ordered in terms of importance for u++...
Some short comments would be good...
I think u++ vs BOOST is also important...

Subject: Re: Materials for articles: "U++ Core comparison to BOOST"
Posted by fudadmin on Sun, 30 Jul 2006 23:24:45 GMT
View Forum Message <> Reply to Message

from u++ homepage
Quote:
NTL is a library of container and algorithm templates. It is designed to solve some problems we see in current C++ standard library STL.

Would it be reasonable to somehow structuraly emphasize NTL part in the list of all Core pieces?

Subject: Re: Materials for articles: "U++ Core comparison to BOOST"
Posted by fudadmin on Sun, 30 Jul 2006 23:27:18 GMT
View Forum Message <> Reply to Message

What are BOOST problems in the eyes of u++?

Subject: Re: Materials for articles: "U++ Core comparison to BOOST"
Posted by fudadmin on Sun, 30 Jul 2006 23:49:03 GMT
View Forum Message <> Reply to Message

Is smart (shared) pointers of BOOST not enough to solve "value transfer semantics"? What advantages have Ptr / Pte? over them?

Subject: Re: Materials for articles: "U++ Core comparison to BOOST"
Posted by fudadmin on Mon, 31 Jul 2006 00:41:10 GMT
View Forum Message <> Reply to Message

fudadmin wrote on Mon, 31 July 2006 00:49Is smart (shared) pointers of BOOST not enough to solve "value transfer semantics"? What advantages have Ptr / Pte? over them?

Or maybe a better question:

Do BOOST and STL "share" a legacy of beeing on average at least 2 times slower than U++ counterparts making some assumptions from:
http://www.ntllib.org/benchmark.html
?

Subject: Re: Materials for articles: "U++ Core comparison to BOOST"
Posted by mirek on Mon, 31 Jul 2006 08:10:08 GMT
View Forum Message <> Reply to Message

fudadmin wrote on Sun, 30 July 2006 19:49Is smart (shared) pointers of BOOST not enough to solve "value transfer semantics"? What advantages have Ptr / Pte? over them?

The difference and the main problem from my view is that

- boost smart shared pointers are "shared"
- boost smart shared pointers are "pointers"

It complicates the design because the entity represents both the pointer AND object and can be owned by more than single entity (something belongs everywhere . You have e.g. to watch carefuly for cyclical references, you have to alter your algorithms so that they work on pointers rather than on object itself.

Often, you never know when instance really is destructed due to "shared" nature of pointer.

Note that Ptr is quite different beast - it is just pointer, does not have any influence of pointee lifetime.

Anyway, Ptr to some degree represents "inverse" problem that U++ has.

In classic GC and also with boost-like smart pointers to some degree, you always know that as long as something points to object, the object itself exists. Means there are no dangling pointers possible. Con in such arrangement is that destructors are not really possible, which leaves you with manual non-memory resources management.

As U++ has taken opposite direction in resource management, using deterministic destructor cleanup, it has potential for dangling pointers. While in most situations this is really not a big trouble (as usually lifetime of pointer does no extent beyond lifetime of pointee), there are a couple of situation, esp. in CtrlCore, where situation is too fuzzy. Therefore Ptr/Pte - pointers that go NULL if pointee dies.

Mirek

Subject: Re: Materials for articles: "U++ Core comparison to BOOST"
Posted by fudadmin on Mon, 31 Jul 2006 09:20:38 GMT
View Forum Message <> Reply to Message

Mirek, has anyone ever done any performance comparisons between u++ and BOOST? If not, what assumtions could be drawn?
Or, from what you've said, is a short short conclusion correct:
1. "while performance u++ vs. BOOST is the same (or very similar?), u++ reduces code, programmers headeaches and program memory.
And (AFAIK,) for big projects smaller memory usage leads to speed improvements."

2. Or "Do BOOST and STL "share" a ANY legacy of beeing on average at least 2 times slower than U++ counterparts"?

P.S.
3. Also, in other words,(as I understand):
in some cases U++ objects behave like very quick "full occupants" and destroy everything what belongs to them (and/or(?) only inside of them?),
in other (which?) cases they can be told by a programmer "be generous, don't care, "they will die themselves"...
That means, more programmable, flexible and managable "spaggetti" relations strengths beetween objects in u++ than BOOST... ?
(I'm trying to find "visual understanding"...)

Subject: Re: Materials for articles: "U++ Core comparison to BOOST"
Posted by mirek on Mon, 31 Jul 2006 10:55:18 GMT
View Forum Message <> Reply to Message

2.: "average" is too strong word here. In some particular cases, U++ containers are much faster (e.g. Vector<String>::Insert), in some very specific cases, they can be slightly (say 5%) slower due to different featureset. The lesson to learn is that you do not pay the price for relative simplicity. And sometimes you can even gain the speed while using simpler to use library.

Mirek

Subject: Re: Materials for articles: "U++ Core comparison to BOOST"
Posted by mirek on Mon, 31 Jul 2006 10:57:19 GMT
View Forum Message <> Reply to Message

...and the legacy STL has is "copy requirement". In other words, STL performs a lot of copies of elements in containers.

Mirek

Subject: Re: Materials for articles: "U++ Core comparison to BOOST"
Posted by fudadmin on Mon, 31 Jul 2006 15:16:48 GMT
View Forum Message <> Reply to Message

luzr wrote on Mon, 31 July 2006 11:57...and the legacy STL has is "copy requirement". In other words, STL performs a lot of copies of elements in containers.

Mirek

And BOOST "sits" on STL with all the consequences ...?

Subject: Re: Materials for articles: "U++ Core comparison to BOOST"
Posted by mirek on Mon, 31 Jul 2006 16:03:40 GMT
View Forum Message <> Reply to Message

Well, there is much more in boost than STL related stuff....

Thinking about it, second STL feature that is not necessary in NTL is iterator concept.

Sure, any container library needs a way how to identify element. The reason why NTL does not need iterators is the invention of Index - associative container with random access.

Now considering boost, there really is a lot of stuff that somehow deals with those two STL features. On one side there are smart pointers that try to defeat copy problem. On other side there

are endless attempts to introduce lambda calculus to easen iterator deadly syntax...

Mirek

Subject: Re: Materials for articles: "U++ Core comparison to BOOST"
Posted by fudadmin on Mon, 31 Jul 2006 19:31:32 GMT
View Forum Message <> Reply to Message

Please have a look at
http://en.wikipedia.org/wiki/Ultimate%2B%2B
and correct any mistakes:
Quote:
Ultimate's "Core" package extends the functionality of C++ with extensive use of templates similarly as Boost libraries.

The base for the fundamental differences between any C++ "container based" libraries, however, is the use and legacy of STL.

There are at least 2 fundamental innovations in Ultimate's "Core" (NTL):
1. "picked behaviour" - architectured "to be in harmony" with STL containers "copy requirement" but to improve performance by avoiding "memory traffic" while copying elements of containers.
2. Index (an associative container with random access) -designed to make STL iterator concept (and syntax) a reduntant balast and to free a programmer from most to it related problems which STL derivative libraries users including Boost libraries are bound to deal with.

Edit:
P.S
And suggest what else to mention/extend...

Subject: Re: Materials for articles: "U++ Core comparison to BOOST"
Posted by mirek on Mon, 31 Jul 2006 21:42:32 GMT
View Forum Message <> Reply to Message

May I rather suggest to start with something else? I do not think boost comparison or history is the most important thing right now.

Having implemented SVG import, you must now understand U++ XML quite well and have it fresh in memory. What about writing nice short tutorial (like the one about NTL or that one unfinished about GUI)?

I think we should rather concentrate on things like this.

Speaking about it, I have established new nest (and assembly) for tutorial examples. So if you would proceed, example packages should be named XML01, XML02 etc...

Mirek

Subject: Re: Materials for articles: "U++ Core comparison to BOOST"
Posted by fudadmin on Mon, 31 Jul 2006 22:09:22 GMT
View Forum Message <> Reply to Message

luzr wrote on Mon, 31 July 2006 22:42May I rather suggest to start with something else? I do not think boost comparison or history is the most important thing right now.

Having implemented SVG import, you must now understand U++ XML quite well and have it fresh in memory. What about writing nice short tutorial (like the one about NTL or that one unfinished about GUI)?

I think we should rather concentrate on things like this.

Speaking about it, I have established new nest (and assembly) for tutorial examples. So if you would proceed, example packages should be named XML01, XML02 etc...

Mirek

Before writing about any GUI or other parts I wanted myself to be "crystal clear" if I understand correctly the fundamentals of U++... and why U++ Ctrls (widgets) are faster than those of other toolkits...
And, I guess, I would also have progressed faster with U++ if I had found easy explanations earlier...

P.S.
Maybe just a few more words for Ultimate's GUI section to wikipedia today?
What do you think?
I'll have to be off-line tomorrow...

Subject: Re: Materials for articles: "U++ Core comparison to BOOST"
Posted by mirek on Tue, 01 Aug 2006 06:14:54 GMT
View Forum Message <> Reply to Message

Then of course it is a good oportunity to discuss. I was afraid that we are gathering resources for "great BOOST U++ comparison article" which is a bit scary idea to me.

As for U++ widgets being faster... Well, first of all, I am not quite sure they are necessary faster

Easier to use, producing more compact and easier to maintain code for large apps, that is the design goal.

But of course, we are trying hard to be fast too. Concerning widgets, the most impact has effective repainting.

Also worth noting is that regular U++ widgets are not implemented as host platform widgets - in other words, for host platform it looks like U++ window is covered by single big widget. That might make things less resource intensive as well (sizeof(Ctrl) is now around 100 bytes, that are all resources needed for widget to exist).

Mirek
