Subject: Question / Issue about Vector
Posted by alexn on Tue, 20 Nov 2007 04:47:39 GMT
View Forum Message <> Reply to Message

Hi,

I'm trying to use the Vector class and within my application, the elements must be aligned on a 16 bytes boundary. (They are being accesses by SSE2 instructions.)

Looking at the code, the Vector template appears to call a global routine that calls "new" for a "byte" element. I would like to avoid rewriting or modifying the standard Vector class.

I am trying to figure out the best way to overwrite the memory allocation routines for the Vector class such that the "vector" was aligned.

In my old (stupid?) view, I would have thought that Vector should have used a "new" based on the type of object. Another approach might have been to add a template argument to Vector that would specify alignment (defaulting to .

I'm quite new to Ultimate++ and and still trying to break my MFC orientation. Any push in the right direction would be appreciated.

Regards,
Alexn

Subject: Re: Question / Issue about Vector
Posted by mr_ped on Tue, 20 Nov 2007 08:45:44 GMT
View Forum Message <> Reply to Message

One ugly quick hack is to allocate bytes (as base class) in Vector, always reserve +15 bytes in total, and use (pointer to 15th element) & (~15) to get aligned start of memory.
But in such case I would probably prefer to not use UPP::Vector at all, and use only new/delete and fixed size ubyte array.

Maybe Mirek will have some more "proper" idea how to make Vector allocation aligned.

Subject: Re: Question / Issue about Vector
Posted by mirek on Tue, 20 Nov 2007 21:45:09 GMT
View Forum Message <> Reply to Message

alexn wrote on Mon, 19 November 2007 23:47Hi,

I'm trying to use the Vector class and within my application, the elements must be aligned on a 16 bytes boundary. (They are being accesses by SSE2 instructions.)

Looking at the code, the Vector template appears to call a global routine that calls "new" for a "byte" element. I would like to avoid rewriting or modifying the standard Vector class.

I am trying to figure out the best way to overwrite the memory allocation routines for the Vector class such that the "vector" was aligned.

In my old (stupid?) view, I would have thought that Vector should have used a "new" based on the type of object. Another approach might have been to add a template argument to Vector that would specify alignment (defaulting to .

I'm quite new to Ultimate++ and and still trying to break my MFC orientation. Any push in the right direction would be appreciated.

Regards,
Alexn

I am afraid there are no guarantees about alignment in Vector and no intend to introduce any. IMO this is way too low-level issue to be reflected in Vector interface contract.

Mirek

Subject: Re: Question / Issue about Vector
Posted by mr_ped on Tue, 20 Nov 2007 23:24:22 GMT
View Forum Message <> Reply to Message

BTW: what about std::vector and your own allocator?
If you, for whatever reason, use the container API extensively enough to make this worth of effort. (I didn't use stdlib extensively enough to be sure, but I think all the standard containers allow to use your own allocators?? Maybe I'm wrong.)

I always liked classic pointer and arrays more than those complex templates, but in PHP in connection with "foreach" I started to appreciate them a little bit. Still whenever I know enough about my array (i.e. fixed size, fixed type, small scope of code using it), I prefer classic [] array without that additional fuss and burden around. Usually in new code I don't know enough, and flexibility of vector allows better refactoring and earlier bug catching.

Subject: Re: Question / Issue about Vector
Posted by alexn on Wed, 21 Nov 2007 06:25:05 GMT
View Forum Message <> Reply to Message

Mirek,

I really like a lot of attributes and the thought processes of Ultimate++ and I want to use as much of the provided classes as possible. One of the things that initially attracted me, was your approach of trying to address the performance issues of std yet provide many of the coding

(methods) and runtime benefits.

Actually, I would think it would be a violation of the Vector class contract if it didn't observe the alignment requirements of the each element. I can not speak for compilers outside of MSVC, as it isn't easy for me to test, but with 7.1 declaring a simple array of an aligned elements delivers the alignment of each element. I struggle to see why the Vector class should be given a free pass.

Before using trying to use Ultimate++, I ended up writing my own "FastArray" class that had many of the positive attributes of the Vector class vs. the standard std::vector class (e.g. moveable), yet providing many of the same methods. I think I actually prefer the Ultimate++ provided methods.

The basic problem I didn't have time to work through was an instrinic way for the class to determine the alignment property of an element when allocating a new block of memory so I defined an optional alignment template attribute that was used by the internal memory allocator. In VC++ I usee the _aligned_memory function to allocate memory. It solved the problem of the address of the first element. sizeof(Element) as the basic increment of memory made sure following elements were aligned.

I think the Vector class is struggling between trying to use the general Ultimate++ heap allocation mechanism that is alignment unaware and adding additional info the the base Vector class to maintain awareness between the allocated address and the aligned address.

I would think that for performance oriented applications this could be an issue. Anyone that tries to take into account the cache alignment of their data structures with the Vector class could be affected by the internal heap manager that assumes an 8 byte alignment is okay for any data element.

IMO, cache and SSE2 alignment is not a low level issue in some number of applications. This becomes even more interesting when trying to move SSE2 based data between UI constructs and the rest of my program.

I can certainly move my old FastArray class over to my Ultimate++ application. I guess I was just hoping to leverage as much of your framework as possible.

Regards,
Alexn

Subject: Re: Question / Issue about Vector
Posted by alexn on Wed, 21 Nov 2007 06:33:34 GMT
View Forum Message <> Reply to Message

Hi,

The basic problem I had with the std was the assumption of the "worse" case classes that needed

individual constructors on moves or expansion. The std allocator would get me the alignment solution but the rest of the class methods would drive performance into the mud.

Yet I wanted some of the runtime checks and protections that std::vector provided. So, although I could use a standard array, I would be consistently writing ASSERTS and Checks... that the code became too "hard to follow".

Hence I ended up a my own class that assumed deep copy, provided alignment and provided my basic requirements of a set of "vector" methods. (each method being added as I needed them.) The only real difference between Vector and my class was alignment awareness.

Ultimate++ assumptions about moveable was one of the "exciting" things about the framework. I really don't want to go back to using std and I would like to leverage Ultimate++ as much as possible.

AlexN

Subject: Re: Question / Issue about Vector
Posted by mirek on Wed, 21 Nov 2007 10:45:12 GMT
View Forum Message <> Reply to Message

mr_ped wrote on Tue, 20 November 2007 18:24BTW: what about std::vector and your own allocator?
If you, for whatever reason, use the container API extensively enough to make this worth of effort. (I didn't use stdlib extensively enough to be sure, but I think all the standard containers allow to use your own allocators?? Maybe I'm wrong.)

AFAIK, especially in the case of std::vector, there are no guarantees provided w.r.t. alignment of elements (other that these are ok for T and normal processing).

Mirek

Subject: Re: Question / Issue about Vector
Posted by mirek on Wed, 21 Nov 2007 10:55:38 GMT
View Forum Message <> Reply to Message

alexn wrote on Wed, 21 November 2007 01:25Mirek,

I really like a lot of attributes and the thought processes of Ultimate++ and I want to use as much

of the provided classes as possible. One of the things that initially attracted me, was your approach of trying to address the performance issues of std yet provide many of the coding (methods) and runtime benefits.

Actually, I would think it would be a violation of the Vector class contract if it didn't observe the alignment requirements of the each element. I can not speak for compilers outside of MSVC, as it isn't easy for me to test, but with 7.1 declaring a simple array of an aligned elements delivers the alignment of each element. I struggle to see why the Vector class should be given a free pass.

Before using trying to use Ultimate++, I ended up writing my own "FastArray" class that had many of the positive attributes of the Vector class vs. the standard std::vector class (e.g. moveable), yet providing many of the same methods. I think I actually prefer the Ultimate++ provided methods.

The basic problem I didn't have time to work through was an instrinic way for the class to determine the alignment property of an element when allocating a new block of memory so I defined an optional alignment template attribute that was used by the internal memory allocator. In VC++ I usee the _aligned_memory function to allocate memory. It solved the problem of the address of the first element. sizeof(Element) as the basic increment of memory made sure following elements were aligned.

I think the Vector class is struggling between trying to use the general Ultimate++ heap allocation mechanism that is alignment unaware and adding additional info the the base Vector class to maintain awareness between the allocated address and the aligned address.

I would think that for performance oriented applications this could be an issue. Anyone that tries to take into account the cache alignment of their data structures with the Vector class could be affected by the internal heap manager that assumes an 8 byte alignment is okay for any data element.

Well, I guess we have spend a lot of time optimizing these issue, only they are considered low-level stuff....

FYI, blocks up to certain size are 16 byte aligned. Large blocks are 8 bytes aligned and are always 16 byte *misaligned* (anyway, for large block, cache-wise, this is not a problem, because these are usually large arrays anyway).

Note: The description of current allocator is in Core/srcimp.

Quote:
IMO, cache and SSE2 alignment is not a low level issue in some number of applications. This becomes even more interesting when trying to move SSE2 based data between UI constructs and the rest of my program.

You are be right, however global 16bytes alignment would break other parameters.

Well, another problem to think about anyway... I am glad that somebody is onboard with

MMX/SSE2 experience.

Mirek

Subject: Re: Question / Issue about Vector
Posted by mirek on Wed, 21 Nov 2007 11:31:48 GMT
View Forum Message <> Reply to Message

Thinking about it, we need "AlignedVector" with another template parameter to defined the alignment.

Mirek
