Subject: Re: Writing Bits object to disk
Posted by crydev on Fri, 28 Apr 2017 17:04:45 GMT
View Forum Message <> Reply to Message

mirek wrote on Thu, 27 April 2017 23:50Problem with this approach is that you have to create intput Vector<bool> argument first, which is likely to spoil any benefits from faster Bits...

Really, this is the issue - to improve speed here, the interface is problem.

Some possible solutions that came to my mind:

void Bits::Set(int pos, int count, dword bits);

Here count <= 32 and you are passing values in bits dword; that would work if you are packing some normal data into Bits.

template
void Bits::Set(int pos, int count, auto lambda /* [=] (int pos) -> bool */)

Here we would provide lambda that returns value for given position - if compiler is good, it should inline well.

tempalte
void Bits::Set(int pos, bool x ...)

Maybe vararg template is a possible solution to the problem too.

Now the question is: How are you using Bits?

Mirek

Thanks Mirek,

I am building a memory scanner. The results of this scanner consist of a set of addresses and a set of corresponding values. The addresses are fairly well structured. That is: they live in pages, and always live at a specific offset from a base address. Therefore, I can efficiently store this data by using Bits. Every bit is an address, and I keep track of the metadata like base address and offsets. However, that means that I have to write billions of bits to the Bits structure. Therefore, my application benefits from vectorized approaches of setting. :)

Having to create an input vector of bools may not be very efficient, but it is more efficient in my test case. Moreover, it is also a nice solution to have a Bits::Set method do this in another way,

not having to do it yourself. :)

I understand that vectorized versions of the Bits::Set function are not portable and should not be in U++ for portability reasons, but instruction pipelining is available on many architectures. I think it can be well exploited in this case!

crydev
