Subject: Re: Writing Bits object to disk
Posted by crydev on Sat, 06 May 2017 08:28:39 GMT
View Forum Message <> Reply to Message

Hi Mirek,

I worked out the test you suggested. The pipelined set function is not faster anymore. I can understand that this is the case. The SSE2 vectorized set function now is around 30% faster:

```
// Non naive vector test :D
const int VectorBoolOrBitsetTestBitSetVectorizedNonNaive(Bits& buffer, const Vector<unsigned char>& rand)
{
 unsigned char vec[32];
 const int count = rand.GetCount();
 for (int i = 0; i < count; i += sizeof(vec))
 {
  for (int j = 0; j < sizeof(vec); ++j)
  {
   vec[j] = rand[i + j] > 50 ? 0x80 : 0x0;
  }

  // Use the vectorized set method.
  buffer.VectorSetNonNaive(i, vec);
 }
 int alloc = 0;
 buffer.Raw(alloc);
 return alloc;
}

// The non naive vector set function, that actually implements only the vector
// setting, and not also the comparison of the input!
void Bits::VectorSetNonNaive(int i, const unsigned char vec[32])
{
 // Check whether i is within the bounds of the container.
 ASSERT(i >= 0 && alloc >= 0);

 // Get the DWORD index for the internal buffer.
 int q = i >> 5;

 // Do we need to expand the internal buffer first?
 if(q >= alloc)
  Expand(q);

 // Get the bit index of the next available DWORD.
 i &= 31;
```

```
 // Create a bitmask with vector intrinsics.
 __m128i boolVecLow = _mm_load_si128((__m128i*)vec);
 __m128i boolVecHigh = _mm_load_si128((__m128i*)(vec + 16));
 const int bitMaskLow = _mm_movemask_epi8(boolVecLow);
 const int bitMaskHigh = _mm_movemask_epi8(boolVecHigh);

 // Set the resulting WORD.
 LowHighDword w;
 w.dw = bp[q];
 w.w1 = (short)bitMaskLow;
 w.w2 = (short)bitMaskHigh;
 bp[q] = w.dw;
}
```

However, I was thinking: if I use vectorized code, I should be allowed to vectorize everything, including the comparison of the input values! I thought about my own use case (where I indeed also have to perform the kind of comparison you suggested), and I realized that this comparison could be vectorized. When I did so, the speeds of the SSE2 and AVX2 vectorized code examples skyrocketed. I understand that you may think of the vectorized comparison as a 'proof-of-concept' rather than a realistic implementation for U++, but I can actually utilize this implementation for my memory scanner. Besides, it was fun to write code like this. I am surprised how easy it can be done and how easily the game is played out.

The source code is of the other tests is once again located at my Bitbucket repository:
https://bitbucket.org/evolution536/cry-performance-test/src/212c3b4f51a29efa0e70841e76cceb11bcaacf06/VectorBoolOrBitsetTest.cpp?at=master&fileviewer=file-view-default

What do you think about a possible 'multi-set' implementation with an interface like:

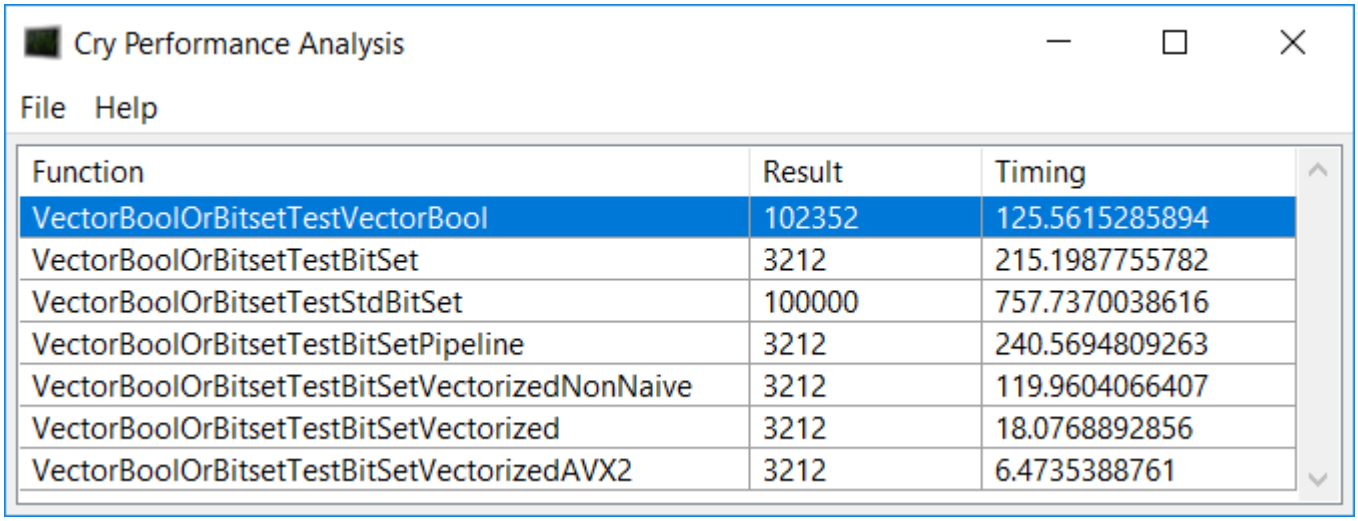
void Set(int i, const Vector<bool>& vec, const int count);

Such function could contain a loop for a vector implementation, and the sequential Set function for the remainder. Maybe also a set function that allows you to manually prepare the dwords and have the set function solely manage the allocation and positioning for you.

crydev

File Attachments

1) Capture.PNG, downloaded 1077 times


Cry Performance Analysis
File Help
Function
Result
Timing
VectorBoolOrBitsetTestVectorBool
102352
125.5615285894
VectorBoolOrBitsetTestBitSet
3212
215.1987755782
VectorBoolOrBitsetTestStdBitSet
100000
757.7370038616
VectorBoolOrBitsetTestBitSetPipeline
3212
240.5694809263
VectorBoolOrBitsetTestBitSetVectorizedNonNaive
3212
119.9604066407
VectorBoolOrBitsetTestBitSetVectorized
3212
18.0768892856
VectorBoolOrBitsetTestBitSetVectorizedAVX2
3212
6.4735388761