Subject: Re: SSE2(/AVX) and alignment issues
Posted by Didier on Fri, 28 Jan 2011 17:43:53 GMT
View Forum Message <> Reply to Message

Hi,

new and delete operators can be overloaded to do what you want.
I already had to do this in my SW for exactly the same reason.

Another solution is to use a <template> replacement new().

I explain myself:

There is the 'placement new()' operator which calls the constructor of an object on a previously allocated memory.
The following code takes advantage of this to create generic allocators/constructors classes. (NB these classes do not need to be instanciated, all methods are static ==> 'typedef' can be used to create custom allocators) !

```
class DefaultAllocator
{
 private:
  DefaultAllocator ( void ) {}
  DefaultAllocator ( const DefaultAllocator& ) {}
  ~DefaultAllocator ( void ) {}
  DefaultAllocator& operator= ( DefaultAllocator& ) { return *this; }

 public:
  static char* alloc ( size_t size )
  {
   return new char[size];  // -------- use the allocator you want -----------
  }

  static void free ( char* buf )
  {
   delete [] buf;         // --------- use the allocator you want -----------
  }
};


template < class ALLOCATOR = DefaultAllocator >
class AllocatorConstructor
{
 public:
  // allocation helper methods
  static void* alloc(size_t size)
```

```
{
 return ( (void*) ALLOCATOR::alloc(size) );
}

template<class BUFFER>
static BUFFER* alloc(void)
{
 return ( (BUFFER*) ALLOCATOR::alloc(sizeof(BUFFER) ) );
}

template<class BUFFER>
static void free(BUFFER* buf)
{
 ALLOCATOR::free((char*) buf);
}

// allocation / construction  methods
template<class OBJ>
static OBJ* allocConstruct ()
{
 OBJ* res = alloc<OBJ>(); // Allocate the memory
 new ( res ) OBJ (); // Call the constructor on allocated memory
 return res;
}


// allocation / construction  methods
template<class OBJ, class PARAM1 >
static OBJ* allocConstruct ( PARAM1 p1 )
{
 OBJ* res = alloc<OBJ>(); // Allocate the memory
 new ( res ) OBJ ( p1 ); // Call the constructor on allocated memory
 return res;
}

template < class OBJ, class PARAM1, class PARAM2 >
static OBJ* allocConstruct ( PARAM1 p1, PARAM1 p2 )
{
 OBJ* res = alloc<OBJ>();    // Allocate the memory
 new ( res ) OBJ ( p1, p2 ); // Call the constructor on allocated memory
 return res;
}

template<class OBJ>
static void freeDestruct ( OBJ* buf )
{
 buf->~OBJ();      // explicit call to destructor
 free<OBJ>( buf ); // free the memory
```

```
  }
};


// ============= USAGE  EXAMPLE ==================
class MallocAllocator
{
 private:
  MallocAllocator ( void ) {}
  MallocAllocator ( const MallocAllocator& ) {}
  ~MallocAllocator ( void ) {}
  MallocAllocator& operator= ( MallocAllocator& ) { return *this; }

 public:
  static char* alloc ( size_t size )
  {
   return (char*) malloc(size);  // -------- use the allocator you want -----------
  }

  static void free ( char* buf )
  {
   free(buf);          // --------- use the allocator you want -----------
  }
};



class Myclass
{
 public:
  int a;
  Myclass() { a = 5; }
  Myclass(int p) { a = p; }
};

// USING 'typedef' you can define system wide definition of you're allocator
typedef AllocatorConstructor<DefaultAllocator> DefaultAllocConst;
typedef AllocatorConstructor<MallocAllocator> MyAllocConst;

void testFct()
{
 Myclass* ptr = DefaultAllocConst::allocConstruct<Myclass>();
 Myclass* ptr2 = MyAllocConst::allocConstruct<Myclass>(10);

 // do you're thing !

 DefaultAllocConst::freeDestruct( ptr );
 MyAllocConst::freeDestruct( ptr2 );
```

```
}
```

This sample class can be easily extended to manage alignement as well (using template parameter ==> 8/16/32... would therefore be set using the typedef.

This is surely can be fit into Upp.
