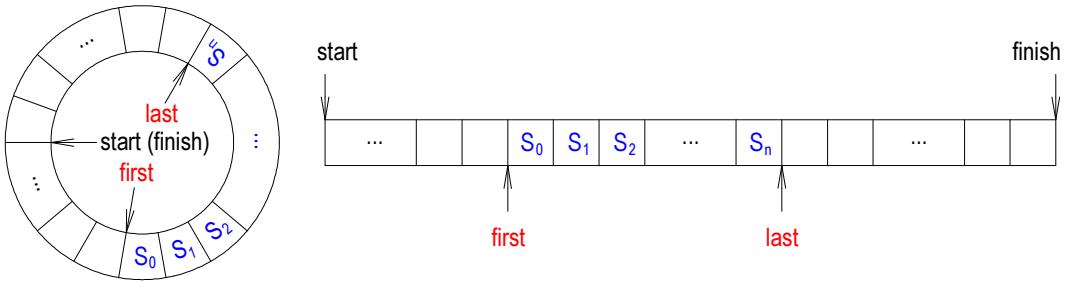
boost::circular_buffer is on the way…

很多时候我们会用到缓冲区或者类似缓冲区的数据结构，如果能事先估计出数据量多大，并尽可能的节约内存，可以使用环形(逻辑上)结构的缓冲区。boost 已经有了一个这样的缓冲区,circular_buffer，由 Jan Gaspar 设计实现，它的数据结构跟传统的环形队列(很多数据结构书上有相关介绍)一样，速度比传统的环形队列快得多。只不过我对它的表现还是不太满意，觉得它还不够快。为此，我设计了一个简单的循环双端队列，它的数据结构与 circular_buffer 没什么两样，但没有编写迭代器，也没有给出太多公有成员函数，只不过它的速度要快一些。下面先来看看它的逻辑结构和物理结构图。



(a) (b)

图(a)是静态双端队列的逻辑结构图，图(b)是对应的物理结构图。循环双端队列有四个指针，指针 start 指向分配内存块的起始地址处，finish 指向该快内存的末尾，first 和 last 是两个自由指针,可以在[start, finish)区间自由移动,在队头添加一个元素时,first 就向左移动一个元素的位置，在队头删除一个元素时，first 就向右移动一个元素的位置，在队尾添加一个元素时，last 向右移动一个元素的位置，在队尾删除一个元素时，last 向左移动一个元素的位置。

开始时，指针 first 和 last 都指向同一个位置(我们的设计是指向[start, finish)区间的正中间)，当循环双端队列满时 last 和 first 之间还剩一个元素的空位(如果不留空位怎么区分队满还是队空？)。比较难处理的问题是指针 first 和 last 移动到队头和队尾怎么办，因为在物理上(在内存中存放元素时)一块存储空间的始末永远都不会构成一个环，环行结构只能在逻辑上出现。

解决此问题的一种方法是取模，例如：first 向左移动一个元素的位置：first = (first - 1 + cap) % cap; last 向右移动一个元素的位置：last = (last + 1) % cap;其中 first 和 last 都是整形变量，cap 是所开辟空间的大小，这就能很好的解决了循环移动指针的问题。这种传统的算法形式上虽然比较简洁，但是速度慢，因为取模需要做除法运算，以现在的 CPU 架构，做一次除法相当于做多次加法。

另外一种解决方案更容易理解，例如：当指针 first 移动到最左端时就让它指向右端，移动到最右端时就让它指向左端，当指针 last 移动到最右端时就让它指向左端，移动到最

左端时就让它指向右端，这种算法速度快，因为做一次判断需要的指令数跟做一次加法需要的指令数相差不多。

为了能够区分何时队满何时队空，循环队列应该至少富裕一个元素的空间，也就是说我们将用 `last + 1 == first` 表示队满，用 `last == first` 表示队空。

为了看看环形缓冲区的性能怎样，做了一个简单的测试，结果如下。

测试用系统配置

操作系统: Windows XP Professional(内核版本 5.1.2600), 英文版

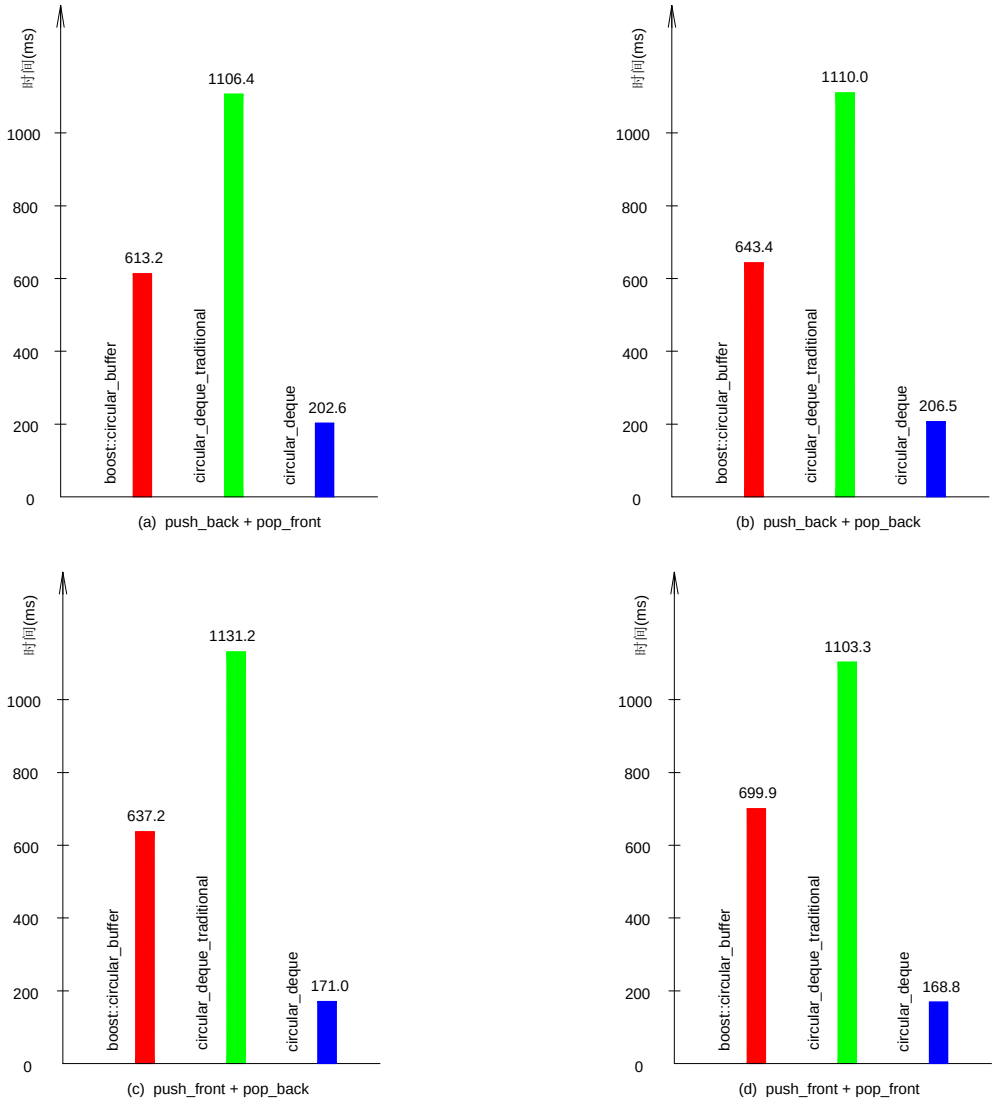
CPU: Intel(R) PIII 1.13MHz, Moblie

编译器: Code::Blocks(svn5616) + MinGW(g++4.4.0)

测试用数据: 32 位的随机数(由 boost::mt19937 产生)

测试数据量: 1 千万，测试时间单位: 毫秒

测试方法: 生成 Release 版(-O3)，运行 5 次取平均值



传统的环形队列速度慢的主要原因是移动指针时需要取模，这很糟糕，因为取模需要做除法，计算机做一次乘法或除法运算比做一次加法耗费的 CPU 周期多很多，由于每移动一次指针都需要做一次取模运算，从而导致整体运行速度大大下降。

boost 1.42 中的 circular_buffer 中 push_front, push_back, pop_front, pop_back 不够快的主要原因是有太多的琐碎操作，这些琐碎的操作会消耗很多 CPU 周期，作者 Jan Gaspar 为何要这样实现，本人不理解，我觉得没有必要那样做(请参考 boost::circular_buffer 的源代码)。

如果您想看看各自的实现，circular_deque 和 circular_deque_traditional 的源代码下载地址