Chapter 6 Boost C++ Libraries

Abstract In this chapter we discuss the Boost C++ API. Boost is a peer-reviewed C++ class library which implements many interesting and useful data structures and algorithms. In particular we discuss the use of Boost smart pointers, Boost asynchronous IO, and IO Streams. Boost also implements many data structures which are not present in the C++ standard library (e.g. bimap). Boost Graph Library (BGL) is presented with the help of real-life example. We compare Boost multi-threading and memory pool performance to APR. We discuss the integration of Python with C++ using Boost. We conclude the chapter with a discussion of Boost Generic Image Processing Library.

Contents

6.1 Boost smart pointer and memory pool .................. 128
6.2 Boost asio framework ................................... 131
6.3 Boost data structures ...................................... 134
6.4 Boost Graph Library ....................................... 135
6.5 Boost Spirit Framework .................................... 138
6.6 Boost multi-threading ..................................... 140
6.7 Boost Python integration ................................... 141
6.8 Boost Generic Image Processing Library (GIL) .......... 141
6.9 Conclusion .................................................. 143

Boost is a collection of C++ libraries and header files which are peer-reviewed, portable, and work well with standard C++ libraries. Indeed, some of the Boost libraries have even become part of the new C++ standard. The main libraries present in Boost (as of version 1.4.2) are given below:

1. Template meta programming and C++ enhancements: these include (i) any, a generic container of single values of different types, (ii) array, STL compliant wrapper for fixed sized arrays, (iii) bimap, bidirectional map for C++, (iv) concept checking tools, (v) foreach in C++, (vi) functional/hash for TR1 C++,
2. Data structures: disjoint sets, date/time, dynamic bitset, property maps, unordered associative containers, universally unique identifier (UUID),
3. Asynchronous IO (asio): see Section 6.2 below,
4. Memory Pool and Smart Pointer: flyweight pattern, memory pool, see Section 6.1 below, pointer container, serialization (see also XDR in Section 9.1.1), smart pointers,
5. Mathematics: linear algebra, quaternions, octonions, interval arithmetic, special functions (see also GNU scientific library, rational number class, uBLAS, in Section 16.5), generic image library,
6. Boost Graph Library (BGL): library for manipulating graphs (vertices and edges),
8. Multi-threading: interprocess, memory mapped files, shared memory, system interface, threading library interface, timer functions, interface to MPI (see Section 12.3.1 for more details on Boost MPI),
9. CRC checksum: see Section 9.2 for a detailed example,
10. Python/C++ integration: see Section 9.7 for an example.

We discuss some of the important Boost C++ libraries below:

### 6.1 Boost smart pointer and memory pool

Memory pools are an important memory optimization technique. The `malloc` provided with the system library has been optimized for best average case performance; in many situations where the programmer is aware of the number, lifetime, usage patterns of objects, it is certainly possible to come up with allocation schemes which either reduce memory consumption, or optimize runtime (or both). We had previously seen the Apache Portable Runtime library (APR) which has a memory pool (see Section 5.4). It is instructive to compare performance of Boost memory pool with that of APR. See Figure 6.1(a) for a runtime comparison. Figure 6.1(b) denotes the number of minor page faults associated with the memory allocation. We will discuss performance optimization tools (perftools) in Section 7.1, and by linking against the provided `libtcmalloc` we reran the experiments. The comparison with and without `libtcmalloc` are shown in Figure 6.2(a), for Boost and Figure 6.2(b) for APR.

The internal memory allocation scheme for Boost and APR is readily apparent from the graphs.

The source code for this example, which can also be used as a short model of using Boost pools and APR is given in Listing 6.1.

```
// \file compare_pool.cpp
// \author Sandeep Koranne, (C) 2010
// \description Example of Boost memory pool class
#include <fstream>
#include <boost/pool/pool.hpp>
#include <boost/pool/object_pool.hpp>
#include <apr_general.h>
#include <iostream>
#include <cassert>
```