Efficient, Portable, Modular C++ Libraries with Applications to OASIS Processing

Patrick LoPresti, RAPID

Abstract – A collection of C++ libraries is presented. Their primary application is efficient processing of OASIS (SEMI P39) data, including decoding, encoding, indexing, and rasterizing. The libraries are sufficiently decoupled and generic to support related applications, and they aspire to be an example of code designed for sharing across divisions.

I. Introduction

OASIS (SEMI P39) [1] has emerged as the dominant interchange format for both wafer and mask design databases. To support OASIS efficiently, and to avoid the expense of commercial EDA software, RAPID has developed a set of libraries and applications for consuming and generating OASIS.

The libraries comprise generic modules useful for database processing in general, including 2D vectors and transforms, a 2D spatial index, polygon raster, and various utility classes to enable efficient handling of files in the hundreds of gigabytes.

The libraries are designed as fully decoupled components, and they are available now to interested parties. They could serve as an early example for how code can be designed and implemented for broad sharing.

II. Design principles

A. *Extensive unit tests*

One key to sharing code between disparate groups is a thorough collection of unit tests. If all relevant use cases are captured in automated tests, code can be improved and shared with confidence.

These libraries were designed with unit testing in mind. They include a wide battery of tests serving both to demonstrate and to enforce their functionality. Contributions of additional tests are welcome and encouraged.

B. *Modularity*

Unit tests are necessary for effective code sharing, but they are not sufficient. Clean, independent interfaces are also important.

This design relies heavily on software decoupling idioms; e.g. the Visitor pattern, dependency injection, and C++ generic programming. The intent is to provide ready-to-use components.

Compiling and linking third-party libraries is often fraught with difficulty. Borrowing an idea from Boost [3], these libraries are "header-only"; to use them, just #include them.

C. *Portability*

To maximize portability, the code generally conforms rigidly to the ISO/IEC 14882:2003 (C++03) standard [2]. Of course, portability and efficiency are sometimes at odds. Where platform-dependent features are required for speed, they have been hidden behind generic interfaces with the implementation isolated to a minimum of files.

D. *Efficiency*

Handling large files was the motivation for these libraries, so efficiency concerns permeate their design.

Memory hierarchy friendliness is essential. All low-level interfaces are carefully designed to require minimal data copying and to permit good memory locality for scaling across many cores.

The low-level geometry library includes a SIMD implementation; see the next section.

III. Library overview

A. Geometry library

The geometry library models a vector space over an arbitrary scalar type. It provides the usual vector operations (addition, dot product, etc.) and several related classes such as bounding boxes, transforms, and a 2D R-tree index.

Speed is particularly important for this low-level library. The primitive classes are lightweight, perform no heap allocations, and are inlined. In addition, for 2D vectors over 64-bit long and double on appropriate platforms, template specializations are provided that use SIMD (SSE) registers and intrinsics. For example, on 2D double-precision vectors, v1 + v2 compiles into a single ADDPD instruction, while union(bbox1, bbox2) compiles into the two-instruction sequence MINPD, MAXPD.

The geometry library provides compile-time adapters for Boost.Polygon and Boost.Geometry, allowing their algorithms to operate directly on this library's data representations.

B. Raster library

RAPID's products have been performing die-to-database inspections for over three decades. Imaging system modeling begins with a raster image of the design database, with edges suitably dithered to permit precise edge placement recovery.

Although RAPID's legacy code for rasterizing figures is very mature, it has two shortcomings: (a) It only handles axis-parallel trapezoids; and (b) it is tightly coupled with the "5xx Database Format", making it difficult to use for other applications.

The "raster" library is a reimplementation of RAPID's existing raster algorithm to alleviate these shortcomings. It handles arbitrary polygons, and it is a completely stand-alone raster engine.

The raster library has been thoroughly tested to ensure it produces identical results to the legacy code, so its output is immediately suitable for modeling applications.

C. OASIS (SEMI P39) library

OASIS is the dominant interchange format for both wafer and mask databases. Designed by committee, it is a rather complex format.

The "p39" library is a toolkit for consuming and generating OASIS that hides the complexity of the format, while remaining low-level enough to support efficient applications.

The P39 Decoder reads OASIS data from an arbitrary source and emits decoded records to an arbitrary destination (visitor object). The Encoder does the reverse. All P39 classes are carefully coded to minimize heap allocations and data copies.

The P39 library includes a multi-threaded engine – called "p39::parallel" – that uses Intel's Threading Building Blocks to enable multi-threaded OASIS applications to be written (relatively) easily.

D. Indexing and Fetching (K39) library

RAPID is adopting an OASIS derivative, dubbed "K39", as its internal database format. K39 is simply OASIS, but with restrictions to enable fast extraction of data clips ("fetching") during inspection.

The K39 library includes code for converting OASIS to K39, serializing and deserializing OASIS clips, generating an index for K39 data, and rapidly fetching a collection of data clips for a set of patches in a swath[1].

E. Miscellaneous library

The "misc" library provides various utility classes. These include clean C++ RAII [4] wrappers around libxml2, libpng, and zlib, as well as custom buffered I/O classes.

The latter enable fast I/O with a generic interface by exposing internal buffers directly to clients. For example, one class performs streaming (i.e. incremental) zlib decompression from an arbitrary data source with a minimum of copies, making it both fast and gentle on the memory hierarchy, while client classes remain oblivious to the decompression details.

IV. Current Applications

A. OASIS analyzer

The "analyze" utility reads and processes every record (cell, figure, etc.) in an OASIS file, calculates precise bounding box and hierarchy information, and produces a summary report.

[1] i.e., a set of roughly-uniform bounding boxes whose aggregate extent is much larger in x than in y

In some ways, this is an extremely simple application. But it is built on the p39::parallel engine, so it provides both sample code for using that engine and perhaps the fastest tool of its kind in the world. Throughput in excess of 1 gigabyte per second, on inputs of hundreds of gigabytes, is routinely obtained.

B. *Index generator*

The index generator produces an on-disk R-tree index for a K39 file, or any OASIS file that roughly complies with K39's restrictions.

Like the analyzer utility, the index generator is built on the p39::parallel engine, and it runs with comparable speed.

C. *K39 Fetcher*

The "Fetcher" is the RAPID system component that acquires and feeds database data to the image computer during an inspection.

The K39 Fetcher relies heavily on the library components. It uses the K39 indexing and fetching code, the P39 decoder, and the serialization provided by the K39 library to create database clips for transmission over the network.

The K39 Fetcher is currently running at customer sites in test engagements.

D. *LPA Raster*

The "LPA" (leaf processing algorithm) is the RAPID system component that processes images to perform defect detection. RAPID's next-generation LPA incorporates the raster engine for OASIS data. This application is currently running at customer sites in test engagements.

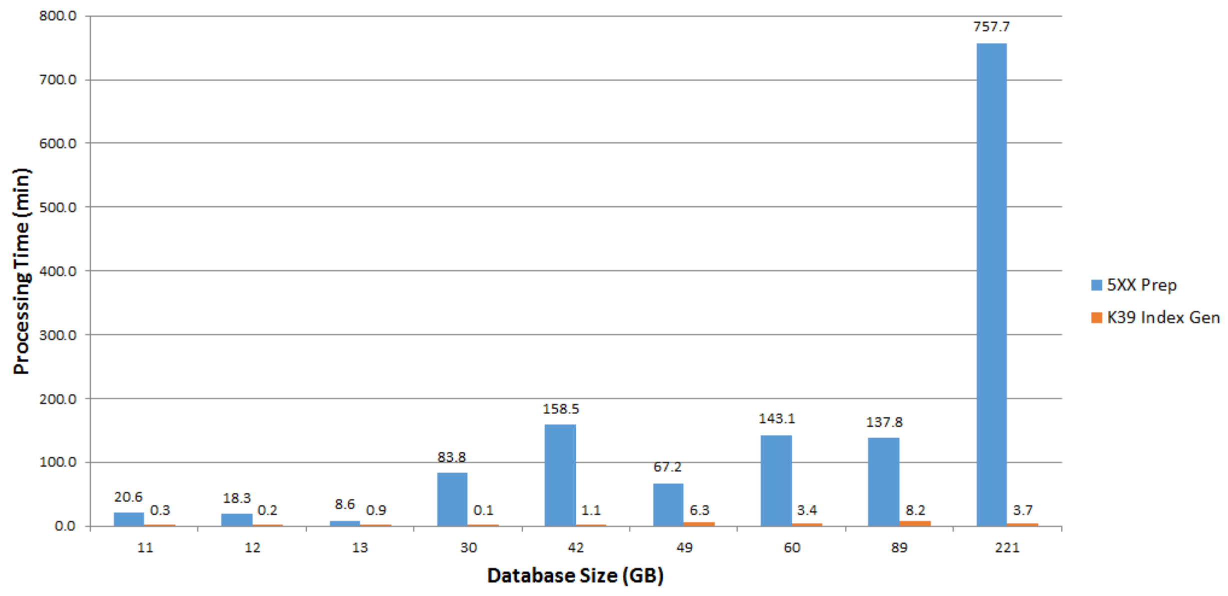
E. *DB Clipping utility*

The "dbclips" utility takes as input an XML file describing a set of desired database clips. It reads the index produced by the index generator, extracts the clips using the K39 fetching code, rasterizes them, and saves the resulting images as PNG files.

The dbclips utility exercises most of the functionality provided by these libraries. It is intended primarily as sample code, not as a production tool.

V. Performance Results

The index generator and LPA raster code are currently running in test engagements at customer sites. The following chart compares, for several customer OASIS databases, the time required for a legacy 5xx database format conversion to that required for K39 index generation.



VI. Conclusion

The need to consume, process, and generate OASIS efficiently, without the cost of commercial EDA tools, inspired the development of a set of C++ libraries useful for database handling in general. Early results show very good performance on large OASIS databases.

The libraries were carefully designed for reuse as well as speed. They are fully decoupled from specific applications and are immediately available to interested parties.

The design and implementation strategies used here could, perhaps, serve as an early example for ways to share code across K-T more broadly.

References

[1] SEMI, "P39-0308," 2008.
[2] ISO/IEC, "Programming languages -- C++," ANSI, New York, 2003.
[3] "Boost," [Online]. Available: http://boost.org/.
[4] "What is meant by RAII?," [Online]. Available: http://stackoverflow.com/q/2321511/.