Creating a Fast FASTA/FASTQ Parser in Perl with XS and kseq.h

Introduction

In bioinformatics, FASTA and FASTQ files are ubiquitous formats for storing sequence data. While there are existing Perl modules for parsing these formats (such as BioPerl's Bio::SeqIO), they can be slow when processing large datasets. In this tutorial, we'll create a high-performance Perl module for parsing FASTA/FASTQ files by leveraging kseq.h, a fast C library by Heng Li.

kseq.h is a lightweight, standalone C library that provides efficient parsing of FASTA and FASTQ formats. It's part of the klib library (https://github.com/attractivechaos/klib), which offers various high-performance bioinformatics tools. By using Perl's XS mechanism, we can create a bridge between Perl and this C library, allowing us to benefit from C's speed while maintaining Perl's ease of use.

This tutorial assumes you have:

- Basic knowledge of Perl
- Basic understanding of C
- Familiarity with the FASTA and FASTQ formats
- A development environment with a C compiler and Perl installed

Understanding FASTA and FASTQ Formats

Before diving into the implementation, let's review the formats we're working with:

FASTA Format

FASTA is a simple text-based format for representing nucleotide or protein sequences:

- Each sequence begins with a header line starting with ">" followed by an identifier and optional description
- The sequence data follows on subsequent lines
- Multiple sequences can be stored in a single file

Example:

```
>seq1 Description of sequence 1
ACGTACGTACGTACGT
>seq2 Description of sequence 2
GCATGCATGCATGCAT
```

FASTQ Format

FASTQ extends FASTA to include quality scores for each nucleotide:

- Each entry consists of four lines:
    1. A header line starting with "@" followed by an identifier and optional description
    2. The sequence data
    3. A line starting with "+" (optionally followed by the same identifier)
    4. Quality scores encoded as ASCII characters

Example:

```
@seq1 Description of sequence 1
ACGTACGTACGT
+
!''*((((***+
@seq2 Description of sequence 2
GCATGCATGCAT
+
IIIIIIIIIII?
```

Understanding kseq.h

kseq.h is a C library that provides fast parsing of FASTA and FASTQ files. It has several important features:

- It can parse both FASTA and FASTQ formats, even mixed in the same file
- It works with gzipped files when used with zlib
- It's extremely fast due to careful buffering and memory management
- It's header-only, making it easy to include in other projects

kseq.h uses a few key data structures:

- `kstring_t`: A dynamic string structure to store sequence data
- `kstream_t`: A buffered stream reader
- `kseq_t`: The main sequence structure that holds name, sequence, quality, etc.

The library is heavily macro-based, which can make it a bit challenging to understand at first, but this is what allows it to be so flexible and efficient.

Setting Up the Module Structure

Let's start by creating the directory structure for our Perl module. We'll name our module `Bio::FASTX::Parser`:

```
mkdir -p Bio-FASTX-Parser/lib/Bio/FASTX
mkdir -p Bio-FASTX-Parser/t
```

```
cd Bio-FASTX-Parser
```

First, let's download kseq.h:

```
curl -o kseq.h https://raw.githubusercontent.com/attractivechaos/klib/master/kseq.h
```

Now, let's create the basic files needed for our module:

1. Makefile.PL

This is the file that will set up the build process for our module:

```
use 5.010;
use strict;
use warnings;
use ExtUtils::MakeMaker;

# Check for zlib
my $zlib_found = 0;
foreach my $path (qw(/usr/local /usr)) {
    if (-f "$path/include/zlib.h" && (-f "$path/lib/libz.so" || -f "$path/lib/libz.dylib'
        $zlib_found = 1;
        last;
    }
}

if (!$zlib_found) {
    warn "Warning: zlib headers and/or library not found. You need to install zlib develo
    warn "For Debian/Ubuntu: sudo apt-get install zlib1g-dev\n";
    warn "For CentOS/RHEL: sudo yum install zlib-devel\n";
    warn "For macOS: brew install zlib\n";
    exit 0;
}

# Write a typemap file
open my $typemap_fh, '>', 'typemap' or die "Could not open typemap file: $!";
print $typemap_fh <<'TYPEMAP';
TYPEMAP
gzFile           T_PTROBJ
kseq_t *         T_PTROBJ
TYPEMAP
close $typemap_fh;

# Define the MakeMaker arguments
WriteMakefile(
    NAME             => 'Bio::FASTX::Parser',
    AUTHOR           => 'Your Name <your.email@example.com>',
    VERSION_FROM     => 'lib/Bio/FASTX/Parser.pm',
    ABSTRACT_FROM    => 'lib/Bio/FASTX/Parser.pm',
    LICENSE          => 'perl_5',
    MIN_PERL_VERSION => '5.010',
    CONFIGURE_REQUIRES => {
        'ExtUtils::MakeMaker' => '0',
    },
    BUILD_REQUIRES => {
        'Test::More' => '0',
    },
    PREREQ_PM => {
        'strict'   => '0',
        'warnings' => '0',
    },
    LIBS => ['-lz'],
    INC => '-I.',
    OBJECT => '$(O_FILES)',
    dist  => { COMPRESS => 'gzip -9f', SUFFIX => 'gz', },
    clean => { FILES => 'Bio-FASTX-Parser-*' },
);
```

This Makefile.PL checks for the zlib library (needed for reading gzipped files), creates a typemap file for the C data types we'll use, and sets up the necessary build parameters.

2. The Perl Module File

Next, let's create the Perl module file:

```
mkdir -p lib/Bio/FASTX
```

Create `lib/Bio/FASTX/Parser.pm`:

```
package Bio::FASTX::Parser;

use 5.010;
use strict;
use warnings;

our $VERSION = '0.01';

require XSLoader;
XSLoader::load('Bio::FASTX::Parser', $VERSION);

1;

__END__

=head1 NAME

Bio::FASTX::Parser - Fast FASTA/FASTQ parser using kseq.h

=head1 SYNOPSIS

  use Bio::FASTX::Parser;

  # Parse a FASTA or FASTQ file (can be gzipped)
  my $parser = Bio::FASTX::Parser->new("sequence.fa.gz");

  # Iterate through all sequences
  while (my $seq = $parser->next_seq()) {
      print "Name: $seq->{name}\n";
      print "Sequence: $seq->{seq}\n";

      # Print comment if available
      print "Comment: $seq->{comment}\n" if exists $seq->{comment};

      # Print quality if available (FASTQ)
      print "Quality: $seq->{qual}\n" if exists $seq->{qual};
  }

=head1 DESCRIPTION

Bio::FASTX::Parser is a Perl module for fast parsing of FASTA and FASTQ files
using the kseq.h library from Heng Li's klib. It supports both uncompressed and
gzipped files.

This module provides a simple interface to access sequences from FASTA/FASTQ files
with high performance and low memory usage.

=head1 METHODS

=head2 new(filename)

Creates a new parser object for the specified file. The file can be either a regular
FASTA/FASTQ file or a gzipped file (.gz extension).

=head2 next_seq()

Returns the next sequence from the file as a hash reference with the following keys:

=over 4

=item * name - The sequence identifier (required)

=item * seq - The sequence string (required)

=item * comment - The comment string (optional)

=item * qual - The quality string for FASTQ files (optional)

=back

Returns undef when there are no more sequences to read.

=head1 AUTHOR

Your Name, E<lt>your.email@example.comE<gt>

=head1 COPYRIGHT AND LICENSE

Copyright (C) 2025 by Your Name

This library is free software; you can redistribute it and/or modify
it under the same terms as Perl itself.
```

```
=cut
```

This file defines the Perl module, loads the XS code using XSLoader, and provides documentation using POD.

Writing the XS Code

Now let's create the XS file that will bridge Perl and C:

Create `Parser.xs` in the root directory:

```
/* FASTA/FASTQ parser using kseq.h */
#include "EXTERN.h"
#include "perl.h"
#include "XSUB.h"

#include <zlib.h>
#include "kseq.h"

// Initialize kseq
KSEQ_INIT(gzFile, gzread)

// Helper function to convert a kseq_t record to a Perl hash reference
SV* kseq_to_hash(pTHX_ kseq_t *seq) {
    HV* hash = newHV();

    // Add name, always present
    hv_store(hash, "name", 4, newSVpvn(seq->name.s, seq->name.l), 0);

    // Add sequence, always present
    hv_store(hash, "seq", 3, newSVpvn(seq->seq.s, seq->seq.l), 0);

    // Add comment if present
    if (seq->comment.l)
        hv_store(hash, "comment", 7, newSVpvn(seq->comment.s, seq->comment.l), 0);

    // Add quality if present
    if (seq->qual.l)
        hv_store(hash, "qual", 4, newSVpvn(seq->qual.s, seq->qual.l), 0);

    return newRV_noinc((SV*)hash);
}

MODULE = Bio::FASTX::Parser   PACKAGE = Bio::FASTX::Parser
PROTOTYPES: DISABLE

SV*
new(class, filename)
    char* class
    char* filename
    CODE:
        gzFile fp;
        kseq_t *seq;
        // Open the file
        fp = gzopen(filename, "r");
        if (fp == NULL)
            croak("Failed to open file: %s", filename);

        // Initialize kseq
        seq = kseq_init(fp);

        // Create a hash to store our object data
        HV* self = newHV();

        // Store the file pointer and seq object as an IV
        hv_store(self, "_fp", 3, newSViv(PTR2IV(fp)), 0);
        hv_store(self, "_seq", 4, newSViv(PTR2IV(seq)), 0);

        // Bless and return
        RETVAL = sv_bless(newRV_noinc((SV*)self), gv_stashpv(class, 0));
    OUTPUT:
        RETVAL

SV*
next_seq(self)
    SV* self
    CODE:
        HV* hash;
        SV** fp_sv;
        SV** seq_sv;
        gzFile fp;
        kseq_t *seq;
        int ret;

        // Get the hash
        if (!SvROK(self) || SvTYPE(SvRV(self)) != SVt_PVHV)
            croak("Not a blessed hash reference");
        hash = (HV*)SvRV(self);

        // Get the file pointer and seq object
        fp_sv = hv_fetch(hash, "_fp", 3, 0);
        seq_sv = hv_fetch(hash, "_seq", 4, 0);

        if (!fp_sv || !seq_sv)
            croak("Invalid object");

        fp = INT2PTR(gzFile, SvIV(*fp_sv));
        seq = INT2PTR(kseq_t*, SvIV(*seq_sv));

        // Read next sequence
        ret = kseq_read(seq);

        if (ret < 0) {
            // EOF or error
            RETVAL = &PL_sv_undef;
        } else {
            // Convert to hash and return
            RETVAL = kseq_to_hash(aTHX_ seq);
        }
    OUTPUT:
        RETVAL

void
DESTROY(self)
    SV* self
    CODE:
        HV* hash;
        SV** fp_sv;
        SV** seq_sv;
        gzFile fp;
        kseq_t *seq;

        // Get the hash
        if (!SvROK(self) || SvTYPE(SvRV(self)) != SVt_PVHV)
            return;
        hash = (HV*)SvRV(self);

        // Get the file pointer and seq object
        fp_sv = hv_fetch(hash, "_fp", 3, 0);
        seq_sv = hv_fetch(hash, "_seq", 4, 0);

        if (!fp_sv || !seq_sv)
            return;

        fp = INT2PTR(gzFile, SvIV(*fp_sv));
        seq = INT2PTR(kseq_t*, SvIV(*seq_sv));

        // Clean up
        kseq_destroy(seq);
        gzclose(fp);
```

Let's break down this XS code:

1. **Header Files and Initialization**:
   - We include the necessary Perl XS headers (`EXTERN.h`, `perl.h`, `XSUB.h`)
   - We include zlib.h for working with gzipped files
   - We include kseq.h, our FASTA/FASTQ parser
   - We use `KSEQ_INIT` to initialize kseq with gzFile and gzread
2. **Helper Function**:
   - `kseq_to_hash` converts a kseq_t record to a Perl hash reference
   - It stores the name, sequence, and optional comment and quality data
3. **XSUBs (XS Subroutines)**:
   - `new`: Constructor that opens a file and initializes the parser
   - `next_seq`: Reads the next sequence from the file
   - `DESTROY`: Destructor that cleans up resources when the object is garbage collected

Writing Tests

Let's create a test file to ensure our module works correctly:

Create `t/01-basic.t`:

```perl
#!/usr/bin/env perl
use strict;
use warnings;
use Test::More tests => 10;
use Bio::FASTX::Parser;

# Create a test FASTA file
my $fasta_file = "test.fa";
open my $fh, '>', $fasta_file or die "Could not create test file: $!";
print $fh <<'FASTA';
>seq1 This is sequence 1
ACGTACGTACGT
>seq2 This is sequence 2
GTCAGTCAGTCA
FASTA
close $fh;

# Test FASTA parsing
my $parser = Bio::FASTX::Parser->new($fasta_file);
ok($parser, "Created parser object for FASTA file");

my $seq1 = $parser->next_seq();
ok($seq1, "Got first sequence");
is($seq1->{name}, "seq1", "Correct sequence name");
is($seq1->{seq}, "ACGTACGTACGT", "Correct sequence");
is($seq1->{comment}, "This is sequence 1", "Correct comment");
ok(!exists $seq1->{qual}, "No quality for FASTA");

my $seq2 = $parser->next_seq();
ok($seq2, "Got second sequence");
is($seq2->{name}, "seq2", "Correct sequence name");
is($seq2->{seq}, "GTCAGTCAGTCA", "Correct sequence");
is($seq2->{comment}, "This is sequence 2", "Correct comment");

my $seq3 = $parser->next_seq();
ok(!defined $seq3, "No more sequences");

# Clean up test file
unlink $fasta_file;

done_testing();
```

This test creates a temporary FASTA file, parses it using our module, and checks that the parsed data is correct.

Building and Testing the Module

With all the files in place, we can now build and test our module:

```
perl Makefile.PL
make
make test
```

If everything goes well, all tests should pass, and we have a working FASTA/FASTQ parser!

Understanding How It Works

Let's look at how our XS module works in more detail:

Memory Management

One of the most important aspects of XS programming is memory management. In our module:

1. We allocate memory in the `new` function when we create the parser object.
2. We increment reference counts (`SvREFCNT_inc`) when necessary to prevent Perl from prematurely garbage collecting objects.
3. We use `newRV_noinc` to create references without incrementing the reference count, which is appropriate when we're transferring ownership to Perl.
4. We properly clean up resources in the `DESTROY` function, which is called when the object is garbage collected.

Data Conversion

Converting between C and Perl data structures is another key aspect:

1. We use `PTR2IV` and `INT2PTR` macros to safely convert between C pointers and Perl integers.
2. We use `newSVpvn` to create new Perl strings from C strings with a known length.
3. We use `newHV` to create Perl hash references, and `hv_store` to add key-value pairs to them.

Error Handling

Proper error handling is essential in XS modules:

1. We check the return value of `gzopen` and call `croak` if it fails.
2. We verify that the `self` parameter is a blessed hash reference before accessing it.
3. We check the return value of `kseq_read` and return `undef` if it's negative (indicating EOF or an error).

Extending the Module

There are several ways we could extend this module:

1. **Add Support for Writing FASTA/FASTQ Files**:
   - Implement functions to write sequences to FASTA or FASTQ files.
2. **Add Support for Indexed Access**:
   - Allow random access to sequences in a file using an index.
3. **Add Support for More File Formats**:
   - Extend the module to support other sequence formats like SAM/BAM.
4. **Optimize for Memory Usage**:
   - Add options to control memory usage, such as limiting the size of sequence buffers.

Conclusion

In this tutorial, we've created a high-performance Perl module for parsing FASTA and FASTQ files using XS and kseq.h. This approach gives us the speed of C while maintaining the ease of use of Perl.

Key takeaways:

- XS allows us to create Perl modules that use C code for performance-critical operations.
- kseq.h is a fast, efficient library for parsing FASTA and FASTQ files.
- Proper memory management is crucial in XS modules to avoid leaks and crashes.
- Well-designed APIs make it easy to use efficient code from high-level languages.

By using this module instead of pure Perl solutions, you can significantly speed up your bioinformatics workflows when working with large sequence files.

References

- kseq.h: https://github.com/attractivechaos/klib/blob/master/kseq.h
- Perl XS documentation: https://perldoc.perl.org/perlxs
- FASTA format: https://en.wikipedia.org/wiki/FASTA_format
- FASTQ format: https://en.wikipedia.org/wiki/FASTQ_format