The Cray BioLib: A High Performance Library for Bioinformatics Applications

James D. Maltby, *Cray Inc*

ABSTRACT: *The new Cray Bioinformatics Library (BioLib) is designed to perform genomic searching, sorting and bit manipulation operations useful in the the analysis of nucleotide and amino acid sequence data. The library makes use of unique Cray vector hardware features and compressed data formats to speed throughput and minimize storage. Though other biological software libraries have been written, this is the first to be specifically designed for high performance computing. Features of the library and a list of calls are included*

.

1. Introduction

The Cray Bioinformatics Library (BioLib) routines perform searching, sorting and low level bit manipulation operations useful in the analysis of nucleotide and amino acid sequence data. These routines are intended to simplify and enable the use of the high-performance features of the Cray vector processors, and was developed on the SV1/ex. The library makes use of unique Cray vector hardware features and compressed data formats to speed throughput and minimize storage. Routines are also available to simplify use of the Solid State Disk (SSD) for the manipulation of extremely large data sets. The routines are designed to be referenced from either Fortran or C, and are currently available on the Cray SV1 series UNICOS systems. A version for the Cray X1 should be available in the second half of 2003.

The BioLib originated in a research project Bill Long was working on with the National Cancer Institute in 2001. They were searching for a particular genetic pattern called a Short Tandem Repeat. Dr. Long was able to formulate the problem in such a way as to take maximum advantage of the bit manipulation functional units built into Cray vector CPUs. By compressing the data into a two-bit-per-nucleotide format and working on large blocks of data at once, very high throughput rates could be achieved, as shown in Figure 1. The routines developed for this project formed the basis of the BioLib, and new features are continually being added.

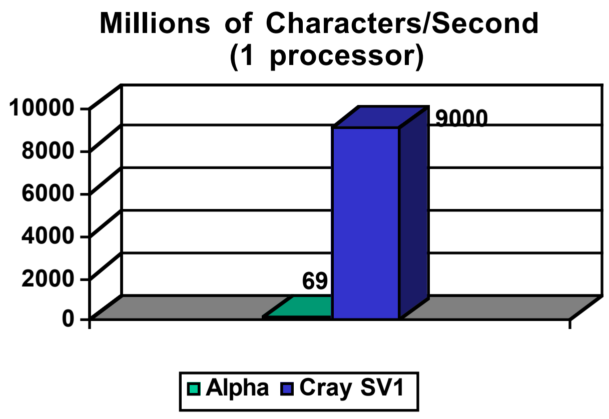

Figure 1. The graph shows the performance advantage of the SV1 over a 667 MHz Alpha processor, searching for a 32 nucleotide sequence in a 34 Mbp database. (Graph courtesy National Cancer Institute)

Over the past year, there has been an ongoing collaboration between the Arctic Regions Supercomputer Center (ARSC) in Fairbanks, Alaska and the Institute for Systems Biology (ISB) in Seattle, Washington to define new modules for the BioLib and exercise it on problems of interest. Though bioinformatics is a very diverse field, it was decided to concentrate the library's functions on genomic searching and manipulation, as well as the support routines necessary for working with very large data sets.

Jim Long of ARSC has developed an open-source version of the BioLib written in C that can be built on virtually any platform. Jim Long has a presentation on the development of the open-source version and performance data at CUG 2003. Though the open source BioLib will not be able to take advantage of the Cray vector hardware, of course, it will be of great utility for biological software development. Applications requiring higher performance or more memory can simply be re-compiled and re-linked on the SV1/ex, and soon on the Cray X1.

2. Other Libraries

There exist several other libraries of functions specifically developed for the biological sciences, but none have the high-performance orientation of the BioLib. The European Molecular Biology Open Software Suite, or EMBOSS, was started in 1997. EMBOSS consists of a large suite of standalone programs in C, designed to solve sequence analysis problems. These programs are designed to be used standalone or as part of custom scripts to solve multi-step problems. They are open-source programs designed to be as portable as possible, and not optimized for any particular architecture.

Another popular package is BioPerl (1995), a collection of Perl scripts for genomic analysis and manipulation. These functions may be combined to form larger scripts for automating common biological computing tasks. Though both these libraries are widely used, they are not oriented towards building high-performance software packages for large data sets. Perl, for example, is usually an interpreted language.

3. Cray Bit Manipulation Features

Since the Cray-1, Cray vector processors have had unique functional units that allow complex bit and logical manipulations at full vector speeds. Typical uses include pattern searching, code manipulations, and now genomic search and comparison. On the SV1/ex, these units include:

Bit Matrix Multiply (BMM)

This is a dynamically programmable bit unit that functions by creating a 64 by 64 array of bits, and performing a matrix-vector multiply on a 64 bit word or entire vector register. This functionality may be used to transpose and manipulate bit subfields within a word, and is crucial to the speed of the compression, decompression and ungapped search routines.

Pop Count / Parity

This function counts the number of one bits in a word, and is used to count the number of "hits" or matches for the search algorithms.

"Snake Shift" (vector register shift)

Using this instruction, an entire vector register can be shifted as a single 4096-bit word. This is very useful for the shift-and-compare algorithm used in the ungapped search routines.

Logical Functions

These include AND, OR, XOR, XNOR, ANDNOT, and MASK. Though not all unique to Cray processors, these functions may be used in combination with the others above in a single vector chime to improve performance.

4. Description of routines

A short description of the available routines follows below, corresponding to the features available in the SV1/ex libcbl version 1.2. More information may be found on the man page intro_libcbl.

Search and Sort routines:

These routines form the core of the library, since sequence search and comparison are some of the most common tasks in genomics.

cb_searchn performs gap-free searches for short sequences of nucleotides, with a specified number of mismatch errors allowed. This is the function shown in Figure 1 above, and is capable of extremely high performance.

cb_repeatn finds exact STRs (short tandem repeats), for repeat lengths from 2 to 16.

cb_sort is a multi-pass sort routine designed to sort large blocks of packed data and return ordered location information for the input data. This can be very useful for decreasing the order of sequence-sequence comparisons.

cb_isort is a parallel sort routine for integer data, using OpenMP parallelization. This allows larger arrays to be sorted with higher performance.

(in the description of the Smith-Waterman routines below, "X" should be substituted with "a" for amino acid searches, "2" for 2-bit nucleotide encoding and "4" for 4-bit nucleotide encoding)

cb_swX_fw calculates the Smith-Waterman score and alignment with full-word accuracy for two input arrays of genomic data. The Smith-Waterman algorithm computes gapped alignments using a dynamic programming algorithm, and has been shown to always yield the optimal alignment. This routine is made up of three routines that may also be called separately, as described below. This offers more flexibility for the programmer, for example to avoid calculating the full alignment until a maximal score has been found.

> **cb_swX_fw_init** initializes the Smith-Waterman scoring matrix and allocates memory.
>
> **cb_swX_fw_align** calculates the optimal alignment corresponding to the maximum score calculated in cb_sw_fw_score.
>
> **cb_sw_fw_score** fills the scoring matrix and returns the maximum score.

Sequence Manipulation routines:

These routines perform common manipulations and scorings of strings of genomic data.

cb_amino_translate_ascii converts nucleotide sequences in ascii format to amino acid sequences, in all three reading frames.

cb_countn_ascii counts the number of A, C, T, G, N characters in an ascii input file.

cb_cghistn creates a histogram of C and G density in a compressed (2-bit) input string, with a user-defined window size. The density of CG characters is an important marker for coding regions.

cb_revcompl generates the reverse complement of a nucleotide string, operating on compressed data.

File Handling routines:

These utilities are very useful for reading the industry-standard FASTA format and converting to the compressed data formats necessary for high performance in the other routines.

cb_read_fasta reads in a multi-record input file in FASTA format.

cb_fasta_convert extracts and organizes data contained in a memory image of a FASTA format data file.

cb_compress compresses nucleotide or amino acid data into various compressed formats.

cb_uncompress converts data in compressed formats back to ascii.

SSD (Solid State Disk) Data Transfer routines:

The SSD can be used to store the extremely large data files required for many genomics problems, and retrieve them at very high speeds. These are high-level interfaces to lower level I/O calls.

cb_ssd_init initializes SSD storage for the other routines.

cb_copy_to_ssd copies a block of data from memory to SSD.

cb_copy_from_ssd copies a block of data from SSD back into memory.

cb_largest_ssdid finds the highest numbered SSD storage area.

cb_ssd_free frees up an SSD storage area.

cb_ssd_errno performs error handling for SSD routines.

Miscellaneous Support routines:

These routines perform memory management and other small tasks that have come up during code development.

cb_malloc allocates memory blocks aligned for highest performance with the other routines (C only; in Fortran use the ALLOCATE statement).

cb_block_zero sets the contents of a block of memory to zero very efficiently.

cb_free frees the memory blocks allocated by cb_malloc (C only; in Fortran use the DEALLOCATE statement).

cb_copy_bits copies a block of bits from one memory location to another; useful for manipulating compressed data.

cb_irand generates a list of 64-bit words with random bit patterns. It can be used to generate random nucleotide sequences.

cb_nmer uses an input string of compressed data to create an array of n-mers, stored one n-mer per machine word.

cb_version provides library version information.

Conclusions

The BioLib provides a new way to accelerate biological code development, providing easy access to the power of the special functional units available on Cray vector processors. This high-performance library differs from other libraries in that it is designed to be used in high-performance C and FORTRAN codes. The soon-to-be released open-source version should ensure wide acceptance of the library and provide a seamless code development and upgrade path.

Acknowledgments

The author would like to thank Bill Long, primary author of the CBL and Jim Long (no relation), author of the open source version. He would also like to thank his colleagues at Arctic Regions Supercomputer Center (ARSC) in Fairbanks and the Institute for Systems Biology (ISB) in Seattle.

About the Author

Jim Maltby is Bioinformatics Practice Leader for North America for Cray Inc. Jim can be reached at Cray Inc. Corporate Headquarters, 411 First Avenue S., Seattle WA 98104. Email: jmaltby@cray.com