The Portable Cray Bioinformatics Library

James Long

CUG, Columbus OH

5/15/2003

The Portable Cray Bioinformatics Library (CBL)

- Introduction
- Portability Issues
- Testing Methodology
- Performance
- Concluding Remarks

Introduction

- High Performance Bio Library
  - Identify the primitives of computational biology
  - Operate on compressed data
- Originally written for Cray SV1
  - Cray version in Fortran/assembly, C callable
  - Uses Cray proprietary hardware
- Portable version written in C
  - Compiles on most unix platforms
  - Use bit-level operations whenever possible, i.e., shift, xor, mask, etc., on compressed data

Introduction

- **Major Version 1.0 primitives**
  - **cb_amino_translate_ascii - translate nucleotides to amino acids, all 3 reading frames**
  - **cb_compress/uncompress - 2, 4, or 5 bit**
  - **cb_copy_bits - copy contiguous sequence of bits, not necessarily word or byte aligned**
  - **cb_irand - generate an array of random words**
  - **cb_read_fasta - load data from FASTA file**
  - **cb_repeatn - find short tandem repeats**
  - **cb_revcompl - reverse complement compressed nucleotide data**
  - **cb_searchn - gap-free nucleotide search w/mismatches**

Portability Issues

- 32/64 bit words
  - Simple, slightly different code for longer length shifts, masks, etc.
- Big Endian, Little Endian
  - Harder, conceptually involves reading and writing left-to-right vs. right-to-left

Big Endian, Little Endian

- **Classic one-word-of-memory definition**

```
Big-endian: Most significant byte in lowest address

      |<--------------word0-------------->|
       byte0    byte1    byte2    byte3
       00100001 00001111 11110000 11111111


Little-endian: Least significant byte in lowest address

      |<--------------word0-------------->|
          byte3    byte2    byte1    byte0
       00100001 00001111 11110000 11111111
```

Big Endian, Little Endian

- **The string “acgta”:**

```
Big-endian:

|<--------word0-------->|<--------word1-------->|<--etc-->|
 byte0 byte1 byte2 byte3 byte4 byte5 byte6 byte7
   a     c     g     t     a   null
```

```
Little-endian:

|<--etc-->|<--------word1-------->|<--------word0-------->|
           byte7 byte6 byte5 byte4 byte3 byte2 byte1 byte0
                       null    a     t     g     c     a
```

Big Endian, Little Endian

- **The string “acgta” compressed:**

```
Big-endian
|<--------------word0-------------->|<--------------word1-------------->|
|byte0---|byte1---|byte2---|byte3---|byte4---|byte5---|byte6---|byte7---|
 01100001 01100011 01100111 01110100 01100001 00000000 <------8-bit ascii
 a        c        g        t        a        null
 00011110 00000000 00000000 00000000   <------ compressed 2-bit string
 a c g t  a null     padded zeros

Little-endian
|<--------------word1-------------->|<--------------word0-------------->|
|byte7---|byte6---|byte5---|byte4---|byte3---|byte2---|byte1---|byte0---|
8-bit ascii -----> 00000000 01100001 01110100 01100111 01100011 01100001
                   null     a        t        g        c        a
compressed 2-bit string ---------->  00000000 00000000 00000000 10110100
                                       padded zeros    null  a  t g c a
```

Testing Methodology

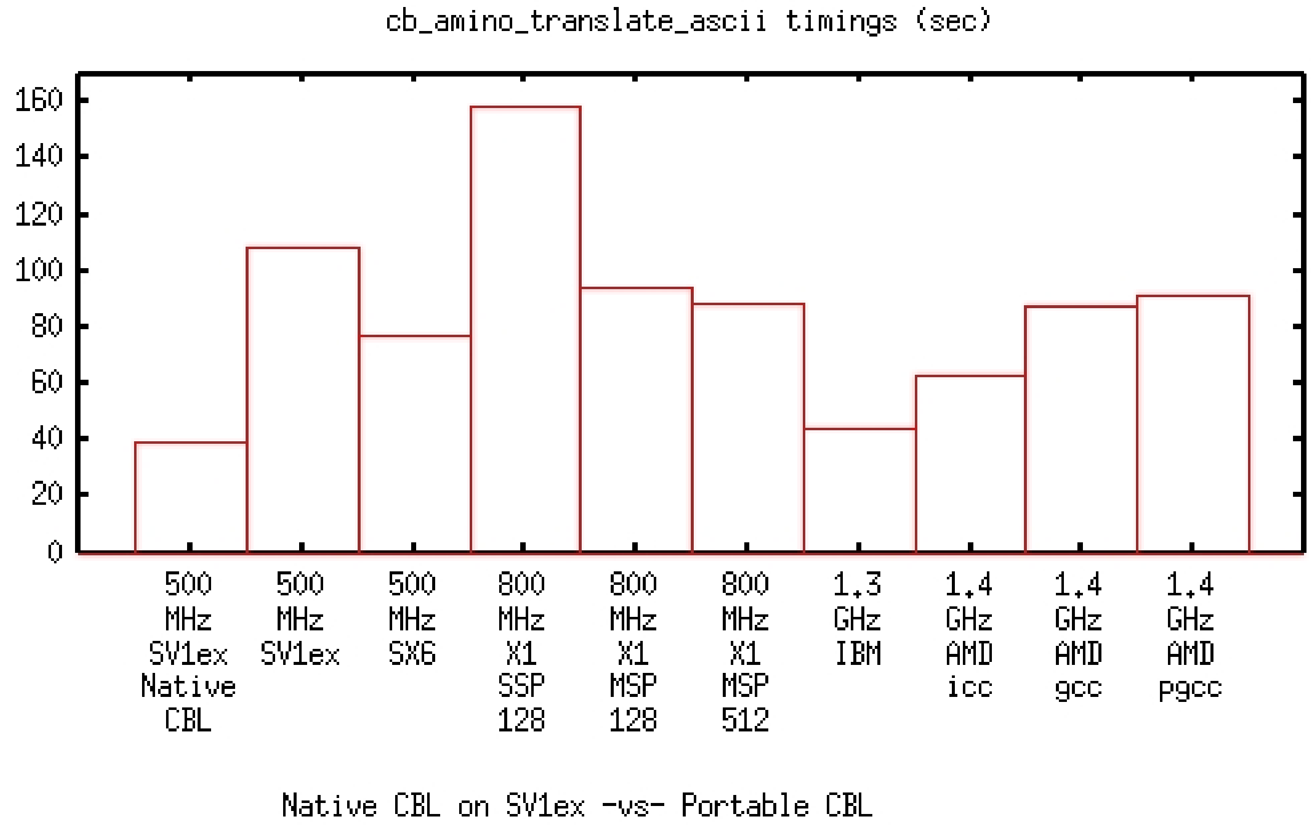
- **XP - Extreme Programming**
  - **write the test first**
    - Simple skeleton to plug in final routine so that a comparison can be made with an unoptimized, easier-to-code routine producing the same output
    - Be sure to compare final vs. unoptimized results across word boundaries and at edges
    - Writing the slower routine first helps clarify issues for the production version

Performance

- Simple benchmarking
    - Combined with test code, 2 loops wrapped around routine
    - Outer loop starts with db length = 256, doubling each time to 33,554,432 (32 MB), extra runs to 512 MB to drive IBM P4 out of L3
    - Inner loop called REP times with varying parameters
    - Inner loop times are summed for final total

cb_amino_translate_ascii

translates nucleotides to amino acids

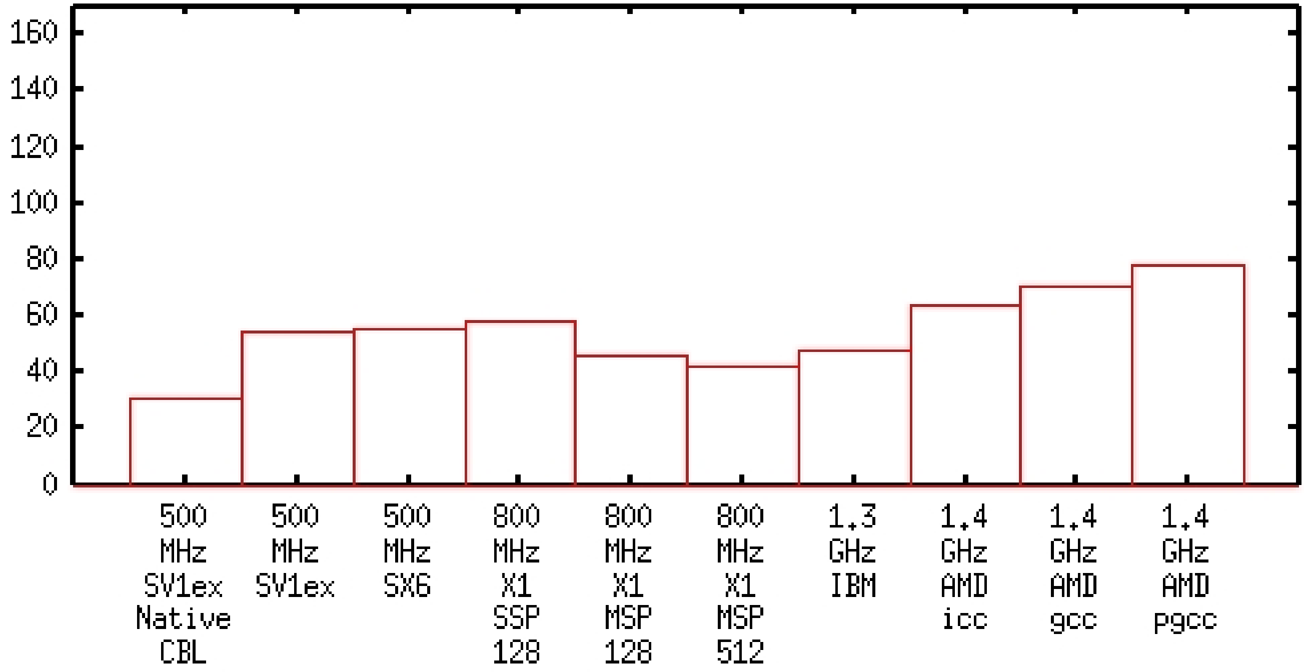


Employs a 64 element **static unsigned long** array as a lookup table to translate groups of 3 nucleotides (compressed in 2-bit mode) into amino acids (in 8-bit ASCII).

cb_compress/uncompress

compresses/uncompresses nucleotides or amino acids from/to ASCII

cb_compress/uncompress timings (sec)

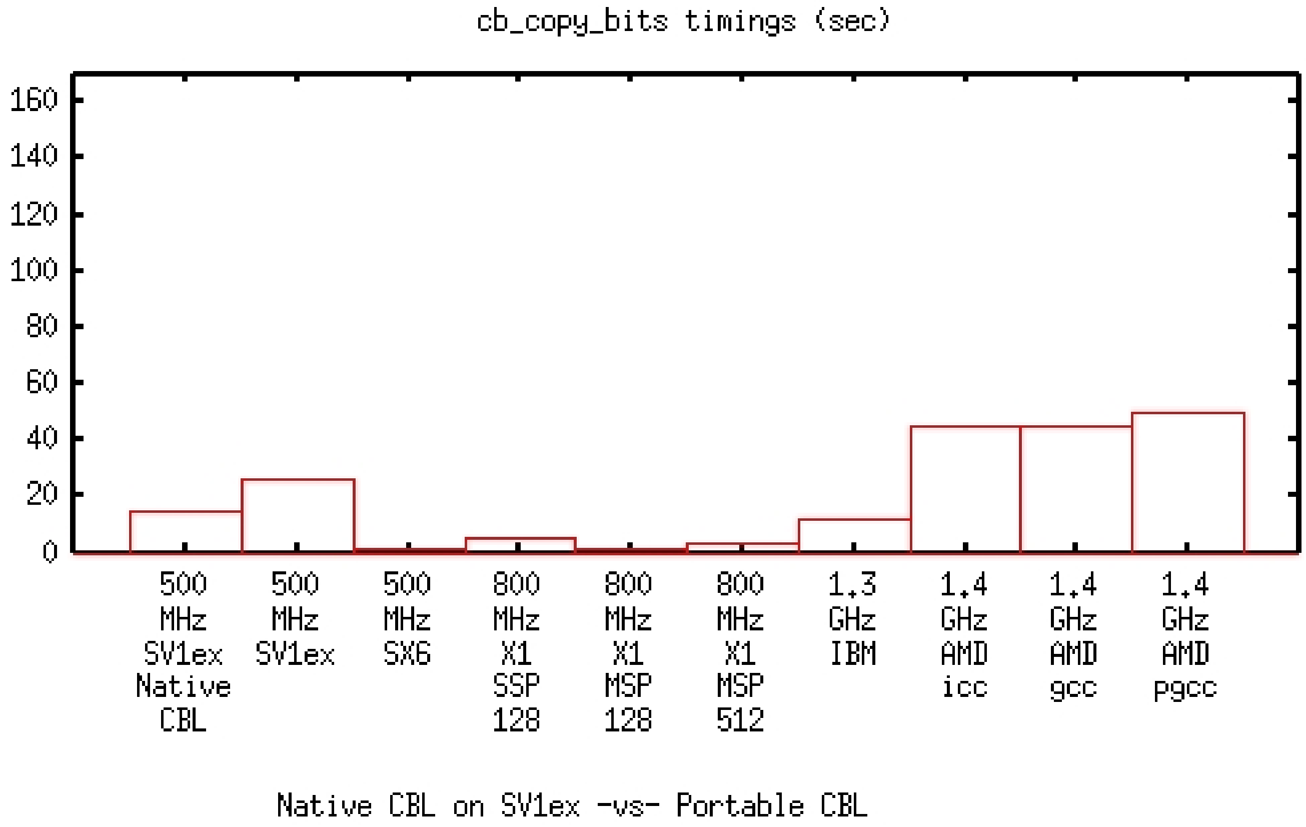


Native CBL on SV1ex -vs- Portable CBL

Consists largely of mask and shift operations, performing well on all platforms.

cb_copy_bits

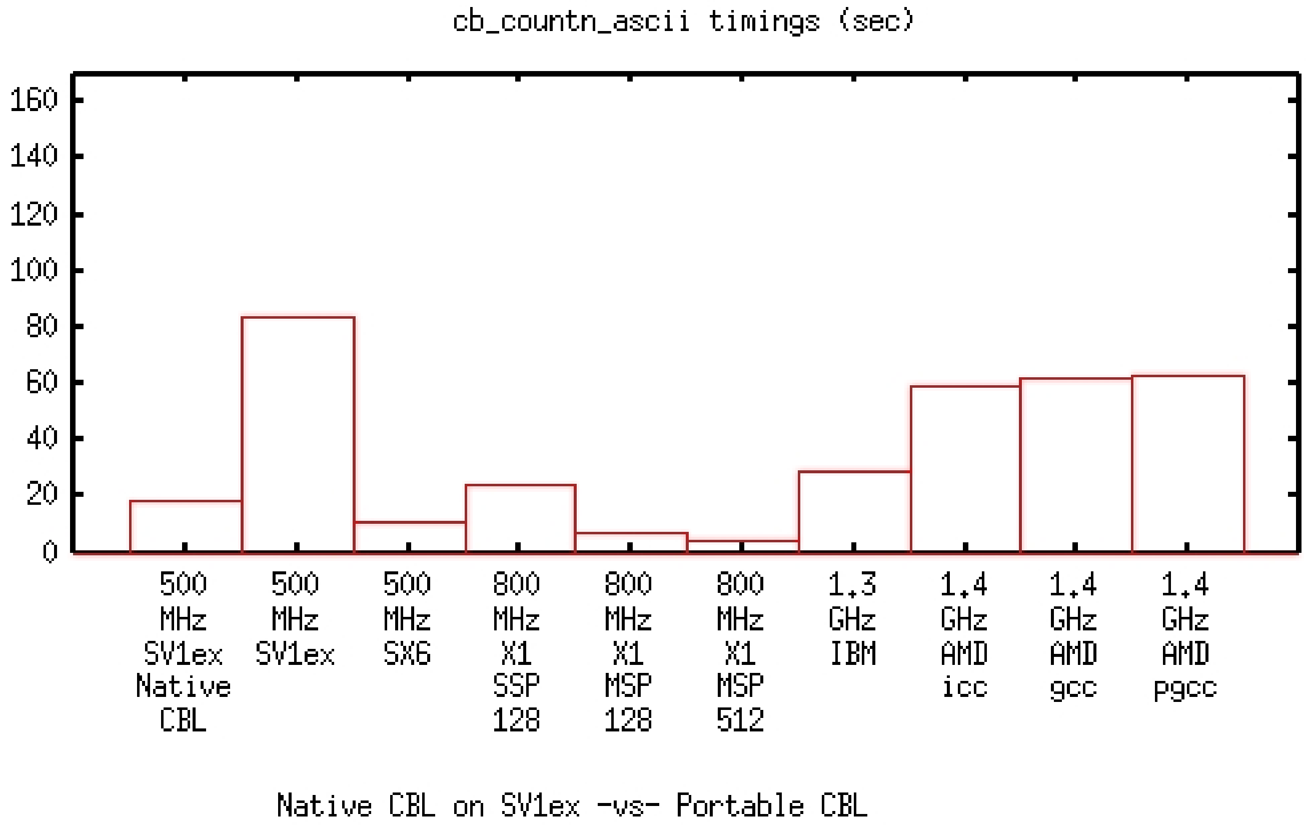
copies a contiguous sequence of memory bits
that is not necessarily word or byte aligned



Performs only a few register operations before moving data back to memory, essentially making this routine a memory bandwidth measure for a platform.

cb_countn_ascii

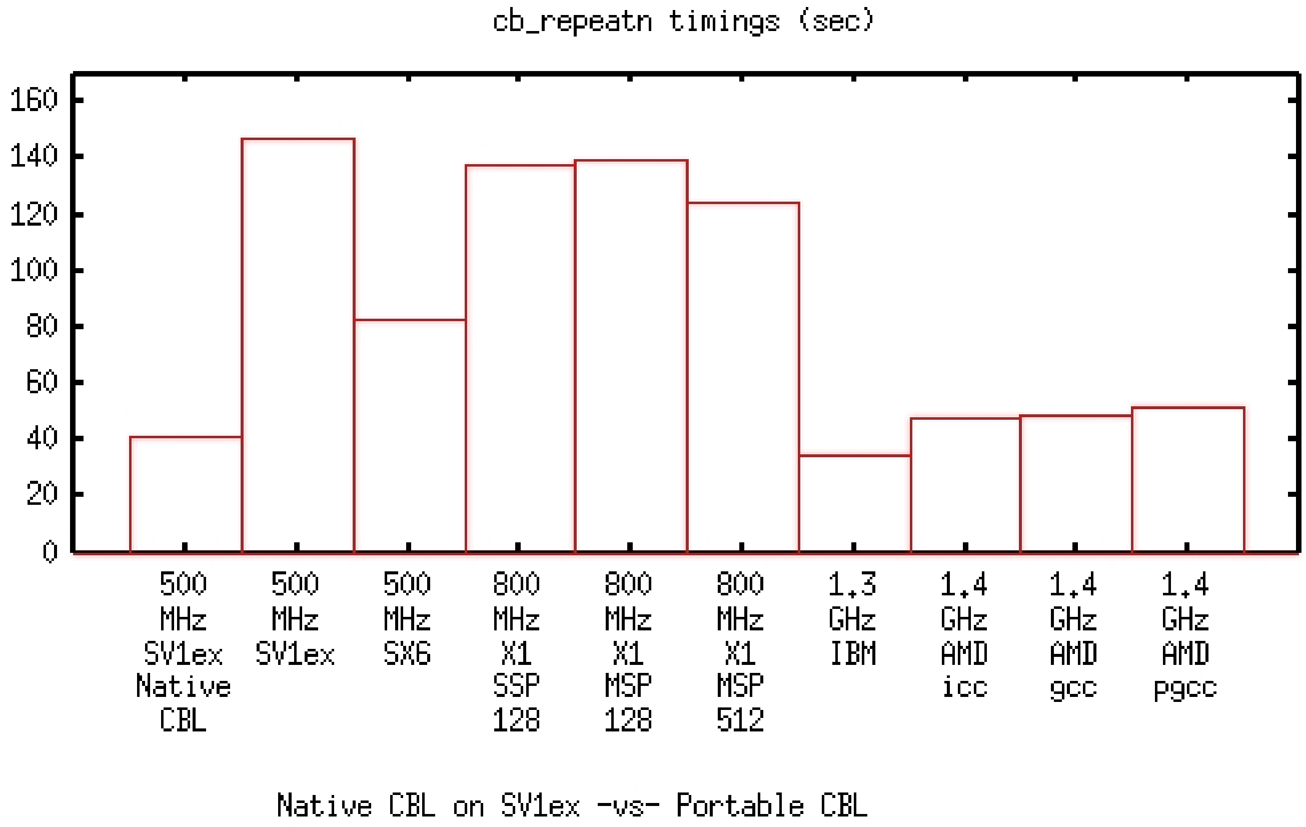
counts A, C, T, G, and N characters in a string



Performs well, SV1ex could be better.

cb_repeatn

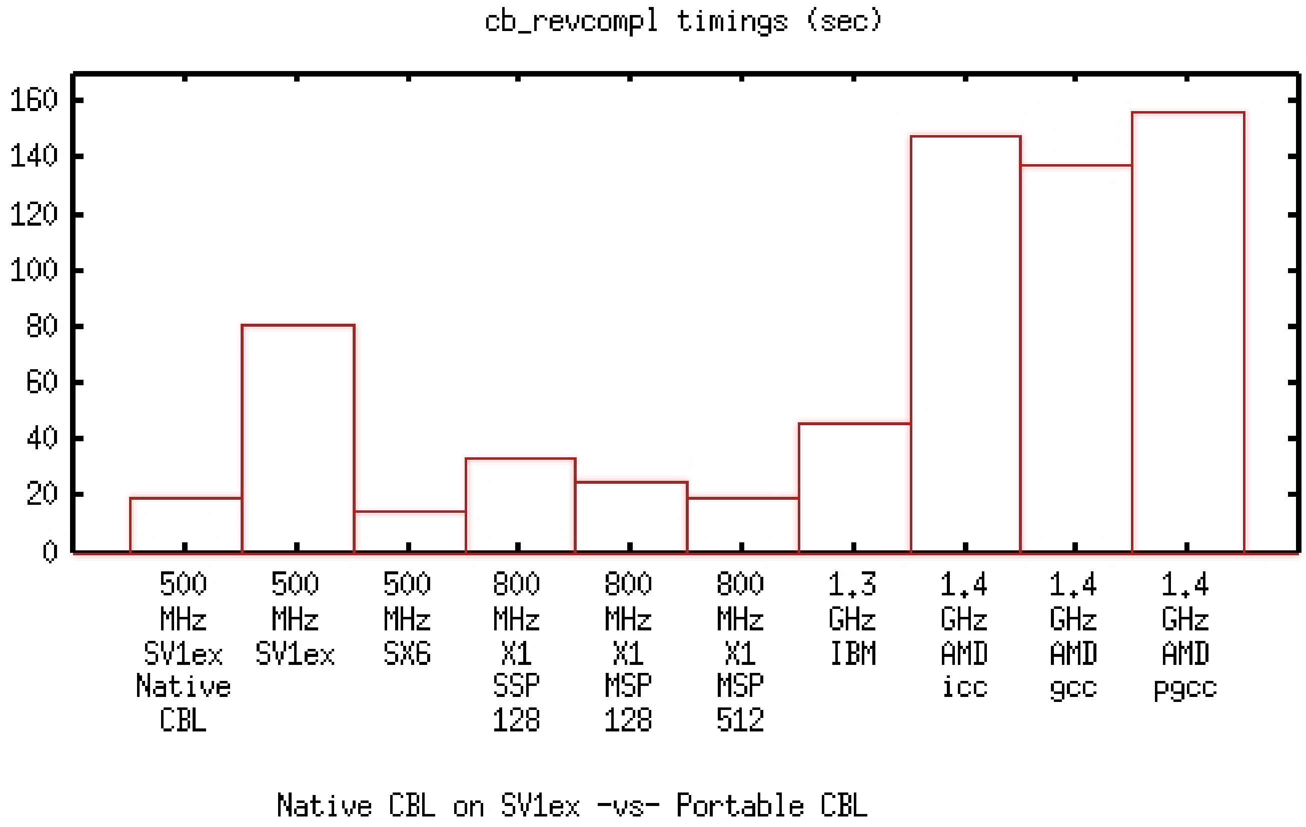
finds short tandem repeats in a nucleotide string



Algorithm needs additional work for vector platforms, but has excellent performance on low-end hardware.

cb_revcompl

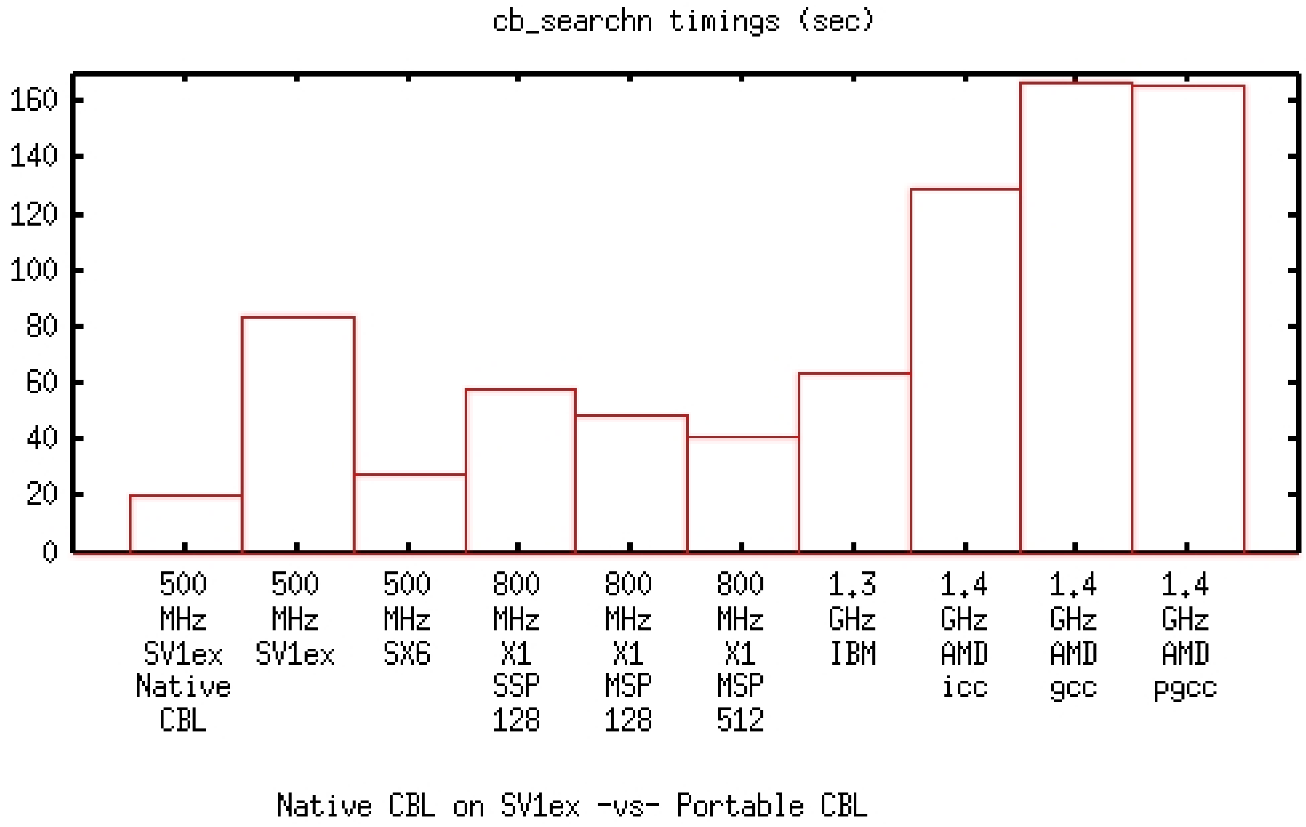
reverse complements compressed nucleotide data



Starts at the end of the database, shifting bits into a new word before a bit reversal within the word followed by a bitwise complement. Needs tuning on low-end hardware.

cb_searchn

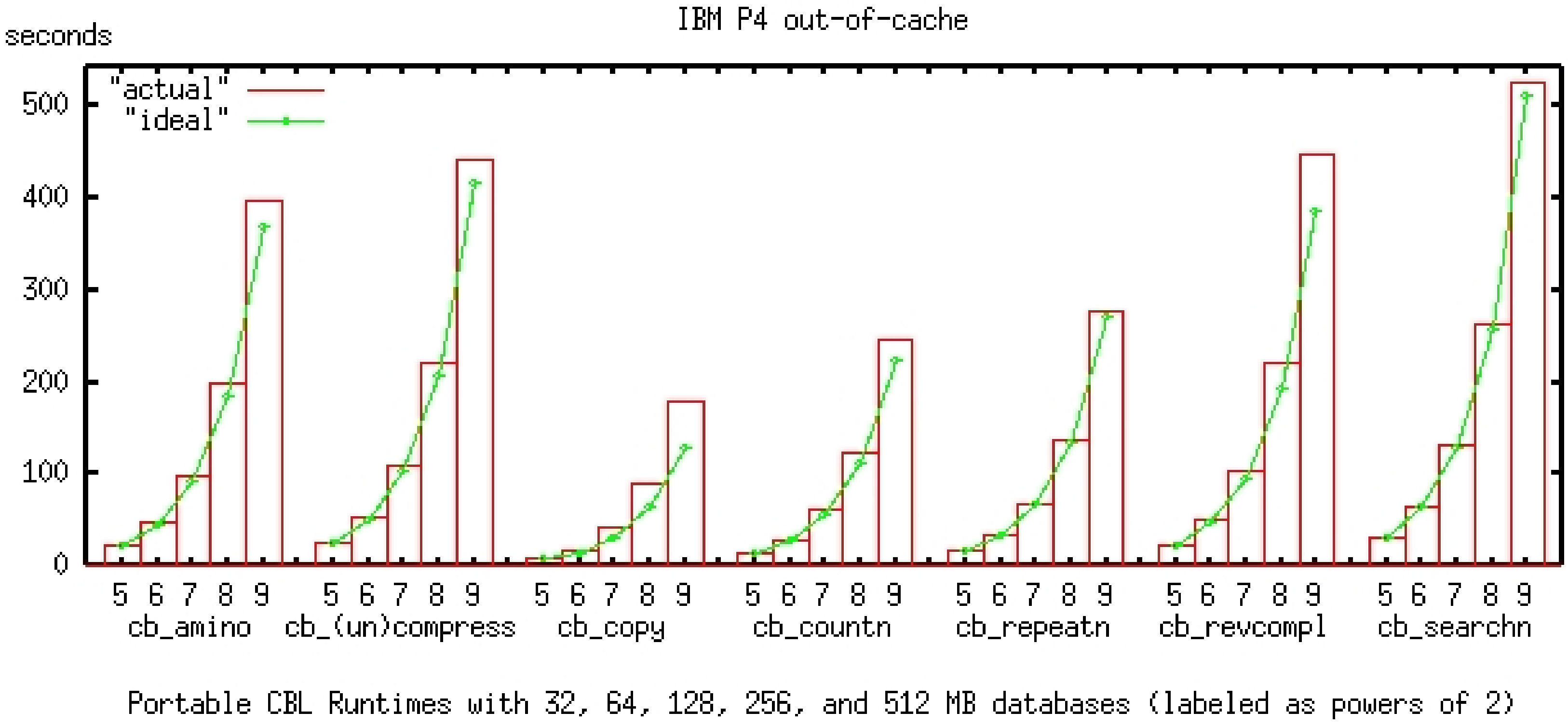
gap-free nucleotide search allowing mismatches



Screens candidates by counting mismatches for only a fraction of each candidate database string, hoping to reject many without having to count all mismatches. Surviving candidates are saved until there are VECTLEN to process. Needs bigger cache on low-end hardware.

IBM P4

32, 64, 128, 256, and 512 MB Databases



IBM P4 processors share a 32 MB L3 cache, large enough to contain the all the databases in the benchmark suite. Compare actual runtimes with ideal runtime computed as doubling the 32 MB time for each routine.

Performance

Integer Benchmarks (seconds)		500		800 MHz X1			1.3 GHz	1.4 GHz		
		MHz		SSP	MSP	MSP	IBM	icc	gcc	pgcc
CBL Function	CRAY	ARSC-SV	SX6	128	128	512	power4	AMD	AMD	AMD
cb_amino_translate_ascii:	39	108	77	158	94	88	44	63	87	91
cb_compress/uncompress:	31	54	55	58	46	42	48	64	70	78
cb_copy_bits:	15	26	1	5	1	3	12	45	45	50
cb_countn_ascii:	18	84	11	24	7	4	29	59	62	63
cb_repeatn:	41	143	83	138	139	124	33	48	49	52
cb_revcompl:	19	81	15	34	25	19	46	148	138	156
cb_searchn:	20	84	28	58	49	41	64	129	167	166

IBM P4:	23,49,99,198,396	26,54,110,222,442	8,18,43,90,178	14,29,61,123,246
	17,35,69,138,277	24,50,105,220,448	32,65,131,262,524	

Concluding Remarks

- Basic things are done very fast
- Library will continue to grow as more primitives are identified
- Portable version is foundation for platform-optimized versions
- Portable version promotes adoption of CBL as a standard
  - Open source is preferred