The 45th CUG Conference
CRAY USER GROUP
FLIGHT TO INSIGHT

The Cray Bioinformatics Library

Jim Maltby

May 15, 2003

Outline of Talk

What is the Cray BioLib?

Cray Bit Manipulation Features

Genomics Examples

BioLib Functionality

What is the Cray BioLib (CBL)?

A library for high-speed genomic manipulation
- – Search and sort routines
- – Sequence manipulation routines
- – SSD data transfer routines

Optimized use of SV1 bit manipulation hardware

Open-source version being developed

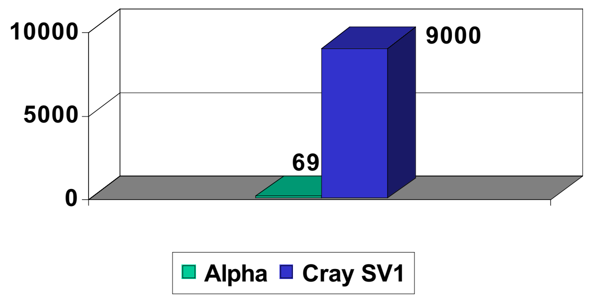
Soon available on X1

Genesis of the Cray BioLib

Cray scientist Bill Long was working with Jack Collins of NCI in 2001.

He discovered special functional units could be used to greatly accelerate genomic search.

Millions of Characters/Second
(1 processor)



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)

Other Biological Libraries

EMBOSS – European Molecular Biology Open Software Suite

- Individual standalone utility programs, also for use in workflow scripts

BioPerl

- Perl scripts for automating common biological computing tasks

→ **Both of these popular libraries are "High level," not necessarily "High performance"**

ARSC / ISB Collaboration

Institute for Systems Biology

Development of open-source, multiplatform version – details in Jim Long's presentation

Large-scale scientific tests planned

Status: Ongoing development

Arctic Region Supercomputing Center

Current CBL status

Cray version 1.2 has recently been released for SV1/ex

– New Smith-Waterman functionality

Multiplatform Open Source version is at v. 1.0

X1 version is planned for 3Q ‘03

Cray Bit Manipulation Features

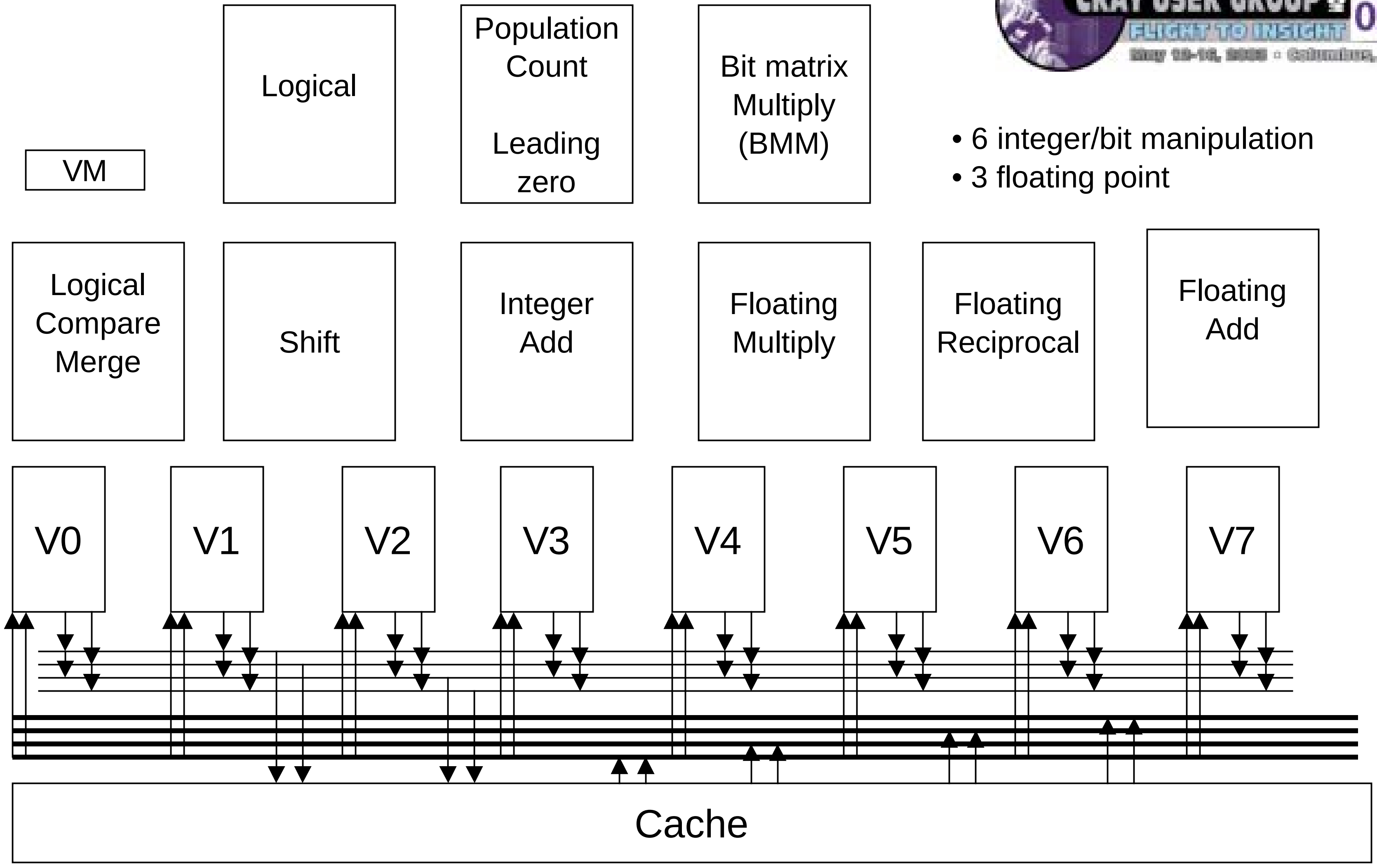
These special features allow complex bit and logical manipulations at full vector speeds.

Typical uses include:

- Pattern searching
- Code manipulations
- Genomic searching and comparison

$\longrightarrow$ **Many are unique to Cray processors**

Cray SV1/ex functional units



- 6 integer/bit manipulation
- 3 floating point

X1 Special Functional Units

Bit Matrix Multiply (BMM)

- Dynamically programmable bit unit

Pop Count

- Counts ones in a word

Leading Zero Count

Copy Sign

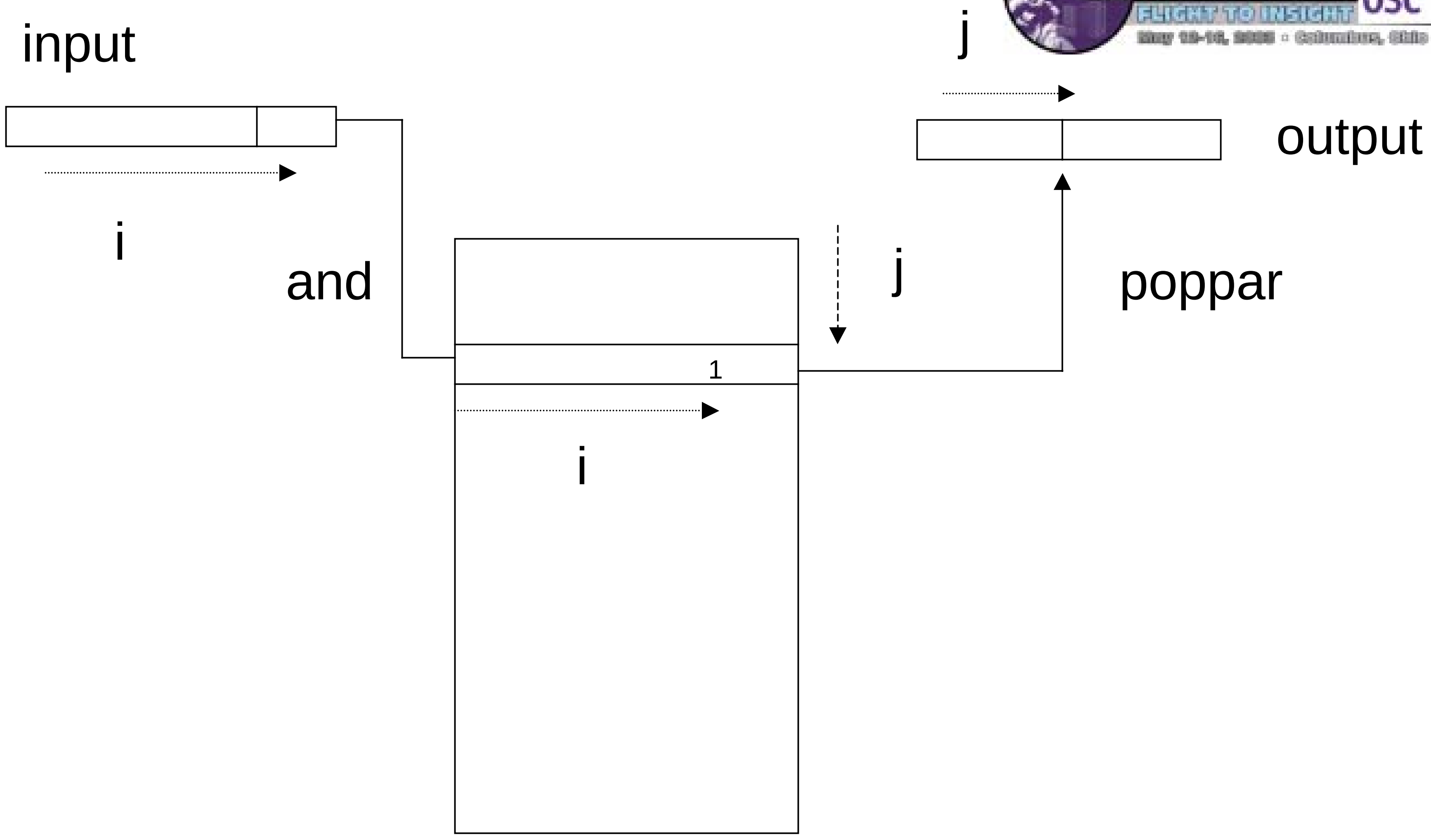
Vector Merge

- Merges registers according to bit mask

Logical

- AND, OR, XOR, XNOR, ANDNOT, MASK

Bit Matrix Multiply



Nucleotide manipulation example

The 45th CUG Conference
CRAY USER GROUP
FLIGHT TO INSIGHT
OSC

ENCODING: A = 00, C = 01, T = 10, G = 11

CAL: v2 v1*BT

Input (v1)

```
A C T G A C C G G T C A ...........C C T
G
A G G C A G G C A G G C...........A G G
C
C C C G T A A C C T G G ...........C C
A G
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

..
.
.
.
.
.
A A A G T C C G T C C G.............C C
C A
```

bmm

```
1000000000000.....................0000
0100000000000.....................0000
0010000000000.....................0000
0001000000000.....................0000
0000100000000.....................0000
0000010000000.....................0000
0000001000000.....................0000
0000000100000.....................0000
1000000000000.....................0000
0100000000000.....................0000
0010000000000...................0000
0001000000000...................0000
0000100000000...................0000
0000010000000...................0000
0000001000000...................0000
0000000100000...................0000
1000000000000...................0000
0100000000000...................0000
.

.
.
1000000000000.....................0000
0100000000000.....................0000
0010000000000.....................0000
0001000000000.....................0000
0000100000000.....................0000
0000010000000.....................0000
0000001000000.....................0000
0000000100000.....................0000
```

Output (v2)

```
A C T G A C T G A C T G ..........A C T G
A G G C A G G C A G G C ........A G G C
C C C G C C C G C C C G ..........C C C G
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

..
.
.
.
.
.
.
.
.
A A A G A A A G A A A G ........... A A AG
```

Search and Sort routines

- **cb_searchn** performs gap-free searches for short sequences of nucleotides, with a specified number of mismatch errors allowed.
- **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 (Fortran only).
- **cb_isort** is a parallel sort routine for integer data, using OpenMP parallelization. This allows larger arrays to be sorted with higher performance.

Smith-Waterman Alignment

(X: a=amino, 2=2 bit, 4=4 bit)

- **cb_swX_fw** calculates the Smith-Waterman score and alignment with full-word accuracy for two input arrays of genomic data. This routine is made up of three routines that may also be called separately, as described below.
- **cb_swX_fw_init** initializes the Smith-Waterman scoring matrix**.**
- **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

- **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.
- **cb_revcompl** generates the reverse complement of a nucleotide string.

SSD routines (SV1/ex)

- **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.

File handling 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.

Bit manipulation routines

– **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.

Miscellaneous support routines

- **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_version** provides library version information.

Conclusions

The CBL provides a new way to accelerate biological code development.

It also provides easy access to the power of the special functional units in the SV1/ex and soon X1.

The Open Source version should ensure wide acceptance.

Thank You!

jmaltby@cray.com

(206) 701-2107