Astronomical Data Analysis Software and Systems XIII
ASP Conference Series, Vol. 314, 2004
F. Ochsenbein, M. Allen, and D. Egret, eds.

LTL - The Little Template Library

Claus A. Gössl, Jan Snigula

Universitäts-Sternwarte München, Scheinerstraße 1, D-81671 München

Niv Drory

Department of Astronomy, University of Texas at Austin, Texas 78712

Abstract. The Little Template Library is an expression templates based C++ library for array processing, image processing, FITS and ASCII I/O, and linear algebra. It is released under the GNU Public License (GPL). Although the library is developed with application to astronomical image and data processing in mind, it is by no means restricted to these fields of application. In fact, it qualifies as a fully general array processing package. Focus is laid on a high abstraction level regarding the handling of expressions involving arrays or parts thereof and linear algebra related operations without the usually involved negative impact on performance. The price to pay is dependence on a compiler implementing enough of the current ANSI C++ specification, as well as significantly higher demand on resources at compile time. The LTL provides dynamic arrays of up to 5 dimensions, sub-arrays and slicing, support for fixed size vectors and matrices including basic linear algebra operations, expression templates based evaluation, and I/O facilities for columnar ASCII and FITS format files. In addition it supplies utility classes for statistics, linear and non-linear least squares fitting, and command line and configuration file parsing. YODA (Drory 2002) and all elements of the WeCAPP reduction pipeline (Riffeser et al. 2001, Gössl & Riffeser 2002, 2003) were implemented using the LTL.

1. LTL Features

1.1. Multidimensional Dynamical Arrays

The multidimensional array class `MArray` features creating and referencing sub-arrays (rank preserving), slicing (rank reducing), e.g. a column of an image, mixing sub-arrays and slices in the same indexing expression (e.g. a sub-matrix of a slice of a cube), referencing the data of other arrays ("views"), and reference counting for the memory chunks holding the actual data. STL-compatible iterators enable interfacing with STL containers and algorithms. `MArrays` resolve arbitrary complex arithmetic expressions without the creation of temporary objects by making use of *expression templates* (Veldhuizen 1995). All standard library math functions are supported, while user supplied functions can be added easily. Indexing arbitrary sets of elements, the evaluation of conditional expres-

© Copyright 2004 Astronomical Society of the Pacific. All rights reserved.

sions, methods for re-indexing, and index iterators are implemented. A set of simple statistical functions (reductions, see Sect. 1.3.) as well as methods for stream, ASCII-file, and FITS file I/O are provided.

1.2. Fixed Vector & Matrix Classes

The fixed vector and matrix classes `FVector` and `FMatrix` provide: compile time fixed size (allows strong optimization), expression template based evaluation of arithmetic and linear algebra expressions, referencing column and row vectors of a matrix, vector dot product, matrix-vector and matrix-matrix dot-product. All operations on small enough objects are automatically unrolled by template meta-programs. In addition there are STL-compatible iterators, and methods for Gauss-Jordan elimination, linear least squares fitting (i.e. polynomial approximation of `MArrays`), and nonlinear least squares fitting (i.e. Marquardt-Levenberg algorithm).

1.3. Simple Statistical Functions

The LTL also provides some statistical functions on `MArrays` (and expressions). These *reductions* include: boolean evaluations (`allof(), noneof(), anyof()`, and `count()`), mean values (average, median, variance, rms, kappa-sigma clipping), and others (like minimum, maximum, sum, product and histogram). All statistical functions may ignore an arbitrary NaN value.

1.4. The Utility Classes

The utility classes provide an easy way of programming a command line or configuration file based user interface. There are also classes for I/O formatting, i.e. date formatting and date conversion etc.

2. Performance

The LTL is explicitly designed to have high performance (i.e. comparable to hand optimized code) while having a high abstraction level to allow the user to write readable and reusable code. Fig. 1 shows for a simple calculation, involving a constant and four two dimensional arrays in single floating point precision, that already for 16×16 elements arrays one line LTL performs as good as hand optimized Code. Our fixed size vectors and matrices can overcome the worse performance for too small sized dynamical arrays.

3. Code Examples

```
MArray<float,2> B = // construct as sub-array
A( Range( A.minIndex(1)+2, A.maxIndex(1)-5 ), Range::all() );
MArray<float,1> C = A(3, Range::all());// construct as slice
// construct from expression
A = exp( sin(100.0 * indexPos(A,2) / M_PI) *
cos(100.0 * indexPos(A,1) / M_PI) );
// replace values via index list
IndexList<3> list = where( A==0 ); A[list] = 1;
```

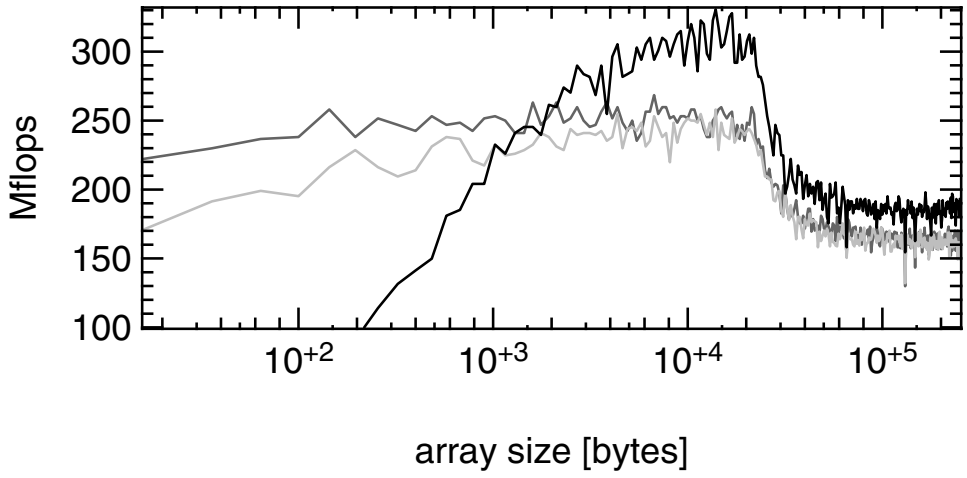

Figure 1. Performance of a standard `C` (light gray), a hand optimized `C` (dark gray), and a `LTL` implementation using dynamical arrays (black) of $A = \text{Constant} * B + C * D$ (A, B, C, D 2-dimensional arrays): Mflops / array size [bytes] of a single array, `GCC-3.1, SPARC / Solaris`

```
// build reciprocal, but avoid division by zero
B = merge( A!=0.0, 1.0/A, 0.0 )

A.setBase(-100, -100); // re-index array
// loop over indices
MArray<float,2>::IndexIterator i = A.indexBegin();
while( i != A.indexEnd() )
{int x = i(1); int y = i(2); ++i;}

// write to / read from stdout / stdin
cout << A; cin >> A;
// read 3rd column from ASCII table foo
AscFile File( "foo" );
MArray<float,1> A = File.readFloatColumn( 3 );

// multiply data of FITS file with value
// of header key GAIN and write back file
FitsIn infile("filename.fits");
MArray<float, 2> A; infile >> A;
const double gain = infile.getFloat("GAIN ");
FitsOut outfile("outname.fits", infile);
outfile.addHistory("multiplied with gain");
A *= gain; outfile << A;

// declare fixed size vectors and matrices
FVector<float, 4> u, v;
```

```
FMatrix<float, 3, 4> A; FMatrix<float, 4, 4> B;
// calculate a dot product
float s = dot(u, v);
// calculate a matrix vector product
FVector<float, 3> w = dot(A, u);
// do a Gauss Jordan elimination
u = GaussJ<float, 4>::solve(B, v);
```

4. Installation

Retrieve the latest versions via CVS:

```
cvs -d :pserver:anonymous@deepthought.usm.uni-muenchen.de:
/usr/share/cvsroot login
Password:  42
cvs -d :pserver:anonymous@deepthought.usm.uni-muenchen.de:
/usr/share/cvsroot checkout ltl
```

If you want to get the latest developer branch use the option `-r ltl-1-7`. Stable releases have even subversion numbers. (Developer branches are odd.) The latest tarballs can be retrieved from
`http://www.usm.uni-muenchen.de/people/drory/ltl/index.html`.

The LTL has been built successfully with `GCC` versions 2.95.2 to 3.3.1 (Linux / IA32, Mac OS X / PPC, Solaris / SPARC), `ICC` version 7.1 (Linux / IA32), Sun C++ version 5.5 (Solaris / SPARC), and IBM Visual Age xlC version 6 (AIX / PPC, Max OS X / PPC). DEC/Compaq/HP (whatever) compiler support is on the way. The LTL's build system is based on GNU `autoconf`.

5. Prospects

We are concentrating on three issues which will be implemented next: A complete documentation using `Doxygen`, support for large files (> 2GB) on all platforms and for FITS extensions, and optimization for CPU floating point vector units.

Acknowledgments. Our thanks are due to Arno Riffeser for extensive testing.

References

Drory, N. 2003, A&A, 397, 371

Gössl C. A. & Riffeser A. 2002, A&A, 381, 1095

Gössl C. A. & Riffeser A. 2003, in ASP Conf. Ser., Vol. 295, ADASS XII, ed. H. E. Payne, R. I. Jedrzejewski, & R. N. Hook (San Francisco: ASP), 229

Riffeser A. et al. 2001, A&A, 379, 362

Veldhuizen T. 1995, C++ Report, Vol. 7 No. 5, pp. 26-31