Boost Basics

• Set of libraries intended to build on the STL
• Peer-reviewed
• Includes:
  • Containers
  • Generic algorithms
  • Anonymous functions (lambdas)
  • C++ to Python bindings
  • Random numbers (all kinds of generators)
  • uBLAS (Basic Linear Algebra Subprograms)
  • “Smart”/“Safe” pointers
• And more
uBLAS : Rationale

- An implementation of BLAS (Fortran):
  - efficient
  - really clean interface
- Uses template-based metaprogramming
  - compile-time polymorphism:
    - slow compile-time, larger binary
    - good run-time performance
    - most of C++'s polymorphism is available
Types (these are all templated)

- (sparse) vector, matrix
- zero_, scalar_, identity_matrix
- triangular_, symmetric_, hermitian_, banded_matrix
- matrix parts:
  - row, column
  - range (part of a row/column)
  - slice (e.g. every 5\text{th} element of a range)
- bi-directional and random access iterators
- STL generic algorithms will work on these
BLAS-style functions

• Ax, Aᵀx
  • like BLAS there are standalone functions to do this, but *, +, etc are overloaded for matrices and vectors anyway
• Specialized products for triangular matrices
• Cholesky etc factorization and other solving tools (lu.hpp – I'm trying to find docs on this)

http://lists.boost.org/MailArchives/ublas/2005/01/0065.php e.g.
Example: Ax=b

```cpp
#include <boost/numeric/ublas/lu.hpp>
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/vector.hpp>

const int n=10;
permutation_matrix<double> P(n);
matrix<double> A(n,n);
vector<double> x(n);
vector<double> rhs(n);

// fill matrix and rhs .. this is pretty straightforward
lu_factorize(A,P);
// Now A and P contain the LU factorization of A
x = rhs;
lu_substitute(A,P,x);
// Now x contains the solution. You could've just put rhs
// in as the 3rd arg if you were okay with overwriting it
```
Performance Notes

http://boost.org/libs/numeric/ublas/doc/overview.htm#rationale

- Benchmarking results are available from that URL, comparing with pure C arrays
  (Peer-reviewed = pure C implementation was reasonable, I assume)
- Ops on small vectors are slower than in C
- Operations on small matrices as efficient
- Ops on large vectors/matrices as efficient
- Tweak preprocessor flags for extra speed
- No comparison with LAPACK(++, etc.)
Using LAPACK with uBLAS

• uBLAS wasn't designed as an alternative to LAPACK, exactly, e.g.
  • LAPACK will solve eigenproblems, etc.
  • uBLAS doesn't have anything that sophisticated
• Some people developed a way of using LAPACK in Boost (uBLAS data structures, etc)
  • http://news.tiker.net/software/boost-binding
• One problem to watch for: uBLAS matrices are row-major by default, and LAPACK expects column-major so you (may) have to convert in order to use LAPACK functions
Stuff built on Boost::uBlas

• Bayes++: Library of Bayesian filters, including Kalman and EKFs

• CGAL: Computational Geometry Algorithms Library

• MIMAS Vision Library (No idea of the quality of this ...)
  http://vision.eng.shu.ac.uk/jan/mimas-docs/index.html

• Adobe Photoshop CS2, etc.

• Seems to be in really wide use, especially uBLAS. More here...