­Parallel tools/Libraries: BLAS

By far the most popular core functionality is provided by BLAS (in the notes). It’s an absolute nightmare to use directly, as it is a FORTRAN implementation, but you import it as a library into C/C++ and then write FORTRAN style code in C/C++… which is somewhere between crazy and idiotic.

The good news is that there’s a parallel implementation of BLAS in the most popular C++ Library, called Boost ([www.boost.org](http://www.boost.org) )

So first, install boost for Linux

sudo apt-get install libboost-all-dev

Note: for reasons beyond my comprehension boost doesn’t always push their updates to repos and God Knows how to install it on a mac:

<https://www.boost.org/doc/libs/1_81_0/more/getting_started/index.html> has a link to their various platform instructions if you want to do it ‘their way’. For version 1.81 but the libraries are still 1.7. This should work with version 1.5x and above.

A program to get you started

|  |
| --- |
| /\*  sudo yum install boost boost-thread boost-devel  from https://www.boost.org/doc/libs/1\_70\_0/libs/numeric/ublas/doc/matrix.html#matrix  \*/  #include <boost/lambda/lambda.hpp>  #include <iostream>  #include <iterator>  #include <algorithm>  #include <boost/numeric/ublas/matrix.hpp>  #include <boost/numeric/ublas/io.hpp>  int main **()** **{**  **using** **namespace** boost**::**numeric**::**ublas**;**  matrix**<**double**>** m **(**3**,** 3**);**  **for** **(**unsigned i **=** 0**;** i **<** m**.**size1 **();** **++** i**)**  **for** **(**unsigned j **=** 0**;** j **<** m**.**size2 **();** **++** j**)**  m **(**i**,** j**)** **=** 3 **\*** i **+** j**;**  std**::**cout **<<** m **<<** std**::**endl**;**  **return** 0**;**  **}** |

That should output something like: [3,3]((0,1,2),(3,4,5),(6,7,8))

Text

Description automatically generated

If not, nothing else is going to work.

1. Make a random array rather than the pre-canned example (the size can be set in the code but you’ll probably want it to stay 3x3 for now, bigger than that will be interesting for testing timing later).
2. Next, figure out how to create a second matrix and to multiply them.

To do that, you’ll need to

#include <boost/numeric/ublas/operation.hpp>

And then find the right kind of product:

<https://www.boost.org/doc/libs/1_81_0/libs/numeric/ublas/doc/operations_overview.html> This is the same for new and old versions.

Note: CAPITAL LETTER VARIABLES ARE MATRICES, lower case are vectors, i,j,k are integers,

1. Calculate the sum of all of the elements of the matrix.

To do that you will need to

#include <boost/numeric/ublas/vector.hpp>

The algorithm is simple but not intuitive: multiply a row vector of 1’s by your matrix, then sum the resulting vector (there is no sum for a matrix).

(so make a scalar\_vector<double>(M.size1()) and product it with your Matrix)

Text

Description automatically generated

1. Ideally, compare to your Lab4 where you did the same basic thing and see what the timing difference is.

**Comparing to lab4, this lab’s code is faster for same 3X3 size of matrices. And lab 5 I was running more than on calculation actually**

Text

Description automatically generated

Sake of useful interest (we’re probably not going to need this but <http://www.netlib.org/blas/blasqr.pdf> is a summary of the BLAS commands)