The Build to Order BLAS User Manual

by Geoffrey Belter, Ian Karlin, Jeremy G. Siek, Thomas Nelson and E.R. Jessup
University of Colorado at Boulder

1. Introduction

The Build to Order (BTO) BLAS compiler takes as input a file specifying a sequence of matrix and vector operations, and creates an implementation in the C language tuned for memory efficiency. The implementation may be serial or parallel using PThreads on shared memory systems. All loop fusion opportunities are enumerated in order to decrease memory traffic. The compiler employs a combination of analytic modeling and empirical testing to quickly make optimization choices and ultimately produce the most memory-efficient subroutine version for the hardware on which the compiler is run.

2. Installation Instructions

Currently, BTO is UNIX-specific because it relies on the gettimeofday() function for timing. The compiler has been used and extensively tested on both Linux and Mac OSX systems.

1. Download the BTO BLAS distribution file **`bto.tar.gz`**.
2. Uncompress and un-archive the distribution file with **`tar -zxvf bto.tar.gz`**
3. To compile BTO BLAS, type **`./install.sh`** in the **`bto`** directory. This creates an executable named **`btoblas`** in the **`bin`** subdirectory. The installation process may take a few minutes; benchmarks are run to determine characteristics of the machine where the compiler will operate. BTO BLAS may make optimization choices based on its machine-specific environment.

3. Creating an Input File

Input to the compiler is a **.m** kernel file including input and output parameters and a sequence of linear algebra operations. These operations are written using the matrix arithmetic syntax of MATLAB. Several such files can be found in the **bto/matlabKernels** subdirectory. Below is an example kernel.

```
BICGK
in
  A : matrix(orientation=column), p :
vector(orientation=column), r : vector(orientation=column)
out
  q : vector(orientation=column), s :
vector(corientation=olumn)
{
  q = A * p
  s = A' * r
}
```

The first line of the kernel is its filename. Next is a list of parameters organized into three sections: **in**, **inout**, and **out**. Each of the three sections is optional. Each section is a list of parameters and their types separated by commas. The **inout** section contains parameters that are both read from and written to during the computation.

Parameters can have the following types: **matrix**, **vector** and **scalar**. For matrix and vector types, specify whether the orientation is **row** or **column**.

The matrix arithmetic section, in curly braces, consists of a sequence of assignment statements. Expressions follow MATLAB conventions.

- A**'** indicates A^T
- ***** indicates multiplication
- **=** indicates assignment from the right-side expression or variable's value to the left-side variable

3.1 Restrictions and Common Errors

Addition and subtraction require both operands to be of the same type and have the same (row or column) orientation. For example, addition of two row-major matrices is possible, but not addition of a row matrix and a column matrix. Likewise, addition of two column vectors is allowed, but adding a column vector and a row vector is not. The result's type must match the operand types.

For example, the result of subtracting two column-major matrices is a

column matrix, not a row matrix. If these conditions are not satisfied, the compiler exits with an error.

The rules of matrix computation concerning multiplication apply here. Let A, B, and C be some arbitrary matrices, and c, d, e scalars.

```
c = d * e
c = row vector * column vector
column vector = column vector * column vector
A = B * C
column vector = A * column vector
```

Matrix-matrix multiply will take any combination of orientations so, for example, multiplying a row-major matrix by a column-major matrix produces a column-major matrix.

The following is an example of a combination that does not make sense according to matrix math rules. The operand dimensions are mismatched. That is, a column vector has dimension (n x 1). If, for example, the computation (n x 1) * (n x 1) is entered, the two inner dimensions, 1 and n, do not match, so the multiplication will not be performed. Multiplying (n x 1) * (1 x n) or (1 x n) * (n x 1) works because the inner dimensions n and n, or 1 and 1, match.

4. Running the Compiler

To run the compiler type **`./bin/btoblas matlabKernels/inputfile.m`** in the bto directory. The compiler takes one mandatory argument as its input, a linear algebra kernel .m file, as described in Section 3. The output will have the same name as the input file, but with suffix **`.c`**. It contains what the compiler's algorithm considers the most efficient C-language version of the kernel. Output lands in the input file's directory location. Optional arguments to **`btoblas`** are described below.

4.1 Options

-h [--help]	Will bring up a help menu with the below options.
-a [--precision] *precision*	Set the precision type for scalars in the generated output (float or double). The default is double.

-t [--threshold] *fraction*	This parameter controls how much empirical testing is performed. Only used with use_model on.
-m [--use_model]	Enable the analytic model when testing kernels. If set the compiler will use a memory model to help with optimization choices. Not recommended for most users.
-e [--empirical_off]	Disable empirical testing. Empirical testing is enabled by default. When empirical testing is disabled, best performance is decided only by analytic model. Disabling the model and empirical testing will generate many versions, but will not attempt to determine the best performing.
-c [--correctness]	Enable correctness testing. Correctness testing is disabled by default. When enabled, the input program is converted to a set of very naive BLAS calls which BTO results are compared to. This requires having and working BLAS with cblas entry points. See "Correctness Testing" section for further details.
-r [--test_param] *start:stop:step*	Set parameters for empirical and correctness testing as: *a:b:n* This directs the tests to use matrix order and vector sizes between start and stop values of *a* and *b* using a step size of *n*. See "Testing" section for additional details.
-b [--backend] *name*	Select the code generator. Two C code generators are currently available. One generates C code with pointers and the other generates C code using variable-length arrays (C99). ptr : generate pointer code noptr : generate VLA code
-s [--search]	Specify the search strategy. One of

search	[ga|ex|random|orthogonal|debug|thread]
--ga_timelimit *arg* (=10)	Number of seconds to run genetic algorithm

4.2 Choosing BTO Option Flags

It is possible to create option combinations that generate unwanted results. The default settings are for routines with operations that contain O(N^2) data and calculations. For routines that contain less data and/or fewer calculations, we suggest using the **-r** option to increase the start and stop values of the calculation modeled and tested. The default *start:stop:step* is 3000:3000:1.

Turning off both empirical testing and the model using the **-e** and **-m** flags results in the compiler being unable to determine which routine version is best. Only one of these flags should be used at a time.

5. Calling the Generated Kernel

The output file consists of one function that implements the specified linear algebra kernel. The name of the C function is the name specified in the input file. The parameters of the C function correspond to the input and output parameters in the following way. For each matrix parameter, there are three parameters to the C function, a pointer to the first element of the matrix, the two integers: the number of rows, and the number of columns. For each vector parameter, there is a pointer to the first element of the vector and an integer specifying vector size. Each matrix or vector must be represented as a contiguous block of memory. The following is the function declaration for the BiCGK kernel given in Section 3.

```
void BICGK(double* A, int A_nrows, int A_ncols,
           double* p, int p_nrows,
           double* r, int r_nrows,
           double* q, int q_nrows,
           double* s, int s_nrows);
```

Some of the size parameters are redundant. For example, in this kernel, the **p_nrows** parameter should have the same value as **A_ncols** because the kernel multiplies matrix **A** by vector **p**. In a future release, the parameter list for the generated function will be more concise. The following is a simple example use of the above **BiCGK** function.

```
#include <stdio.h>

void BICGK(double* A, int A_nrows, int A_ncols, double* p,
int p_nrows, double* r, int r_nrows, double* q, int q_nrows,
double* s, int s_nrows);

int main(int argc, char* argv[])
{
  double A[] = { 1, 1, 1,
                 1, 1, 1,
                 1, 1, 1 };
  int m = 3;
  int n = 3;
  double p[] = { 1, 2, 3 };
  double r[] = { 4, 5, 6 };

  double q[] = { 0, 0, 0};
  double s[] = { 0, 0, 0};
  int i;

  BICGK(A, m, n, p, n, r, m, q, m, s, n);

  printf("q: ");
  for (i = 0; i != m; ++i)
    printf("%f ", q[i]);
  printf("\n");

  printf("s: ");
  for (i = 0; i != n; ++i)
    printf("%f ", s[i]);
  printf("\n");
}
```

The output of this example is
 q: 6.000000 6.000000 6.000000
s: 15.000000 15.000000 15.000000

6. Testing

BTO BLAS has an empirical tester which measures execution time using **`gettimeofday()`**, and a correctness tester which can be enabled to verify generated code. The empirical tester is enabled by default and assists in the process of identifying the best set of optimizations. The correctness tester is disabled by default, but may be enabled with the command line

flag **-c**.

Both of these tests have a set of user-definable features including compiler, compiler flags, libraries, and include paths. These values are set in the file **bto/make.inc**. The 'compilers' and 'flags' features are shared by both tests. These are set in **make.inc** as **TCC** and **TFLAGS**. Any other compilation command line flags specific to either test can be set using **CORRECT_INC** or **EMPIRIC_INC**.

Sizes used in these testers is set for each run of BTO using the **-r** flag. This flag requires *start*:*stop*:*step* to be specified. The use of these sizes is explained in the following subsections for each type of test.

6.1 Empirical Testing

Empirical testing is enabled by default. This test can be disabled using the command line flag **-e**.

Beyond the compilation controls found in make.inc, a user should specify appropriate size information using the **-r** flag. The default *start*:*stop*:*step* for the **-r** flag is 3000:3000:1, which which will empirically test using square matrices of order 3000 and vectors of size 3000. The default setting will give valid empirical times for many matrix-vector operations, but may not be large enough to result in a 'good' BTO-BLAS run for certain vector-vector operations, or very powerful machines. In a case like this, simply increase the *start* and *stop* values.

Currently, the empirical tester only handles square matrices. All vector sizes and matrix orders are the same. The empirical tester uses the largest available size for determining best performance. For example if **-r 500:1000:500** is specified, the empirical test will be run for sizes of 500 and 1000, but only the performance from 1000 will be used. For compilation speed it would be better to use **-r 1000:1000:1**.

The empirical tester is available in a temporary working directory. If BTO BLAS has run on the kernel **PATH/bicg.m**, the empirical tester is located in the subdirectory **PATH/bicg_tmp/bicgETester.c** .

6.2 Correctness Testing

Correctness testing is disabled by default. This test can be enabled using the command line flag **-c**. Enabling this test requires that a BLAS library with cblas entry points be included in **bto/make.inc** using **CORRECT_INC**. If a publicly available BLAS is present on a machine, it is likely that the

following will work.

```
CORRECT_INC = -I./ -lblas
```

(In some cases cblas must be specified explicitly)
```
CORRECT_INC = -I./ -lcblas -lblas
```

(If BLAS is located in the user's home directory)
```
CORRECT_INC = -I./ -L$(HOME)/BLAS/ -lcblas -lblas
```

The input program is converted to naive cblas calls. The sequence of calls is not an ideal use of BLAS, and performance comparison against the correctness tester would not be a fair comparison to BLAS.

When the correctness tester is turned on, it only analyzes what the empirical tester would otherwise. This can be controlled using the **-t** flag. Setting **-t 1** will test all versions produced. If **-t** is not used, model settings and differences in input programs will determine what is sent to the correctness tester.

Beyond the settings in **make.inc**, the user should specify appropriate size information using the **-r** flag. The correctness tester will look at sizes within the specified range, including rectangular matrices. The following is an example call to BTO.

```
./bin/btoblas matlabKernels/bicg.m -t1 -c -r 25:100:25
```

This will test all versions produced for the kernel **bicg.m**, using combinations of the sizes 25, 50, 75, and 100. The combinations are determined based on what the input program is. Given a simple input such as this which performs one computation, adding two matrices, there will be two common sizes, M and N. The test generator will look like the following.

```
for (M = 25; M <= 100; M += 25)
        for (N = 25; N <= 100; N += 25)
                perform test
```

The correctness tester is available in a temporary working directory. If **PATH_TO_KERNEL/bicg.m** has been run, the correctness tester **bicgCTester.c** for the kernel **bicg.m** lives in the subdirectory **PATH_TO_KERNEL/bicg_tmp**.

7. Further Information

Publications concerning BTO BLAS are available on our website **ecee.colorado.edu/wpmu/btoblas/**. If you wish to cite our work or are looking for a more detailed explanation of some aspect of it, the following papers are a relatively recent and comprehensive overview of the project.

Details about the serial compiler, model and how the two interact:

Geoffrey Belter, Ian Karlin, Elizabeth Jessup, Jeremy Siek. **Automating the Generation of Composed Linear Algebra Kernels[pdf].** In *SC09: the International Conference on High Performance Computing, Networking, Storage, and Analysis*. November, 2009.

Information concerning parallel code generation:

G. Belter, J. G. Siek, I. Karlin and E. R. Jessup. **Automatic Generation of Tiled and Parallel Linear Algebra Routines: A partitioning framework for the BTO Compiler.** In the Fifth International Workshop on Automatic Performance Tuning (iWAPT'10), Berkeley, CA, June 2010, pp. 1-15.

Information on the parallel memory model:

I. Karlin, E. R. Jessup, G. Belter and J. G. Siek. **Parallel Memory Prediction for Fused Linear Algebra Routines.** In the 1st International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computing Systems (PBMS 10), New Orleans, LA, November 2010, pp. 1-8.

Appendix

A. Syntax and Grammar

The syntax for parameters is given by the following grammar.

```
NAME  = [a-Z][a-Z0-9]*
orientation ::= "row" | "column"
type ::= "scalar" | "matrix(" orientation ")" | "vector("
orientation ")"
parameter ::= NAME ":" type
parameter_list ::= parameter | parameter "," parameter_list
```

The specifications for matrix arithmetic are enclosed in braces and consist of a sequence of assignment statements. Expressions follow MATLAB conventions with ' for transpose, * for multiplication, and the variable on the left side of the equal sign (=) is assigned the result of the calculation on the right side. The grammar for the subset of MATLAB employed by

BTO matrix kernels is shown below.

```
NUM = [0-9]+
body ::= "{" stmt* "}"
stmt ::= NAME "=" expr
expr ::= NUM
         | NAME
         | expr "+" expr
         | expr "-" expr
         | expr "*" expr
        | "-" expr
         | expr "'"
         | "(" expr ")"
```

The grammar for the entire input file is as follows.

```
file ::= NAME ["in" parameter_list] ["inout" parameter_list]
["out" parameter_list] body
```