1st BLIS Retreat. Austin (Texas)

BLIS Hands-on Session

September 4, 2013

Cheatsheet

BLIS Webpage: `http://code.google.com/p/blis/`

Obtaining BLIS: `git clone https://code.google.com/p/blis/`

Building BLIS:

1. Create a new configuration (or use reference one)
2. `./configure <configuration>`
3. `make`
4. `make install`

Outline

1. Requisites. Remote access

2. Building BLIS
   - Step 0: Obtaining BLIS
   - Step 1: Framework configuration
   - Step 2: `make` configuration
   - Step 3: Compilation
   - Step 4: Installation

3. Linking to BLIS

4. Building the BLIS Test Suite

5. Optimizing BLIS

Hands on

- Two options:
  1. Use your GNU/Linux laptop, or
  2. Remotely access `quatro.csres.utexas.edu`

On your laptop. Requisites

- GNU/Linux or UNIX-like system
- GNU Bash 2.0, GNU make, working C compiler
- GNU Octave if you need to create performance plots

On quatro.csres.utexas.edu

- We have setup a guest account for each assistant. Please ask for your username and password
- `ssh USER@quatro.csres.utexas.edu`
- Intel i7-930 - 24 Gb RAM

Outline

1. Requisites. Remote access

2. Building BLIS
   - Step 0: Obtaining BLIS
   - Step 1: Framework configuration
   - Step 2: `make` configuration
   - Step 3: Compilation
   - Step 4: Installation

3. Linking to BLIS

4. Building the BLIS Test Suite

5. Optimizing BLIS

Obtaining BLIS

1. Get a copy of BLIS: `git clone https://code.google.com/p/blis`
2. The top level directory should look like:

```
$ ls
CHANGELOG  INSTALL  Makefile  build   configure  kernels  version
CREDITS    LICENSE  README    config  frame      test
```

Overview

- Generally, a framework configuration consists of:
    1. A few key **header files** with important definitions
    2. **Makefile definitions** with compiler and compiler flags
    3. **Optimized kernels**, typically specified via a symbolic link (optional)

- Configuration files reside inside a subdirectory in the `config` directory.
- You can use the `reference` directory as a template:

```
$ ls config/reference
bli_config.h  bli_kernel.h  make_defs.mk
# Use the reference configuration as a template to create an x86_64opt configuration.
$ cp -r config/reference config/x86_64opt
$ ls config/x86_64opt
bli_config.h  bli_kernel.h  make_defs.mk
```

bli_config.h

- Specify some general parameters of the BLIS configuration
- For example, properties of memory allocator
- Should be auto-descriptive

bli_kernel.h

C preprocessor macros associated with kernels and microkernels

Kernel blocksizes. If you are only concerned with level-3 operations, focus on cache and register blocksizes

Kernel definition. You have to set **ONE** definition per operation. BLIS prepends `s`, `d`, `c`, `z` to create a typed function instance. For example, GEMM_UKERNEL can be defined as follows:

```
#define GEMM_UKERNEL gemm_ref_4x4
```

Kernel naming. You **MUST** name each kernel datataype according to the following convention:

```
void bli_s<name>( <parameter list> );
void bli_d<name>( <parameter list> );
void bli_c<name>( <parameter list> );
void bli_z<name>( <parameter list> );
```

where `<name>` is the name defined by GEMM_UKERNEL above

bli_kernel.h (Cont.)

Kernel location. You **MUST** add a symbolic link to the directory where your kernels reside. For example, to use kernels for the x86_64 architecture provided with the distribution:

```
$ pwd
/home/field/google_code/blis/config/x86_64opt
$ ls
bli_config.h  bli_kernel.h  make_defs.mk
# Look at which kernel sets are available.
$ ls ../../kernels
x86  x86_64
# Symbolically link to x86_64 kernel directory.
$ ln -s ../../kernels/x86_64 kernels
$ ls
bli_config.h  bli_kernel.h  kernels  make_defs.mk
# Make sure the symlink looks correct.
$ ls -l kernels
lrwxrwxrwx 1 field dept 20 Dec  1 18:13 kernels -> ../../kernels/x86_64
```

make_defs.mk

- Contains general make definitions
- E.g. compiler, compiler flags, . . .
- These definitions are inherited by the test/ and testsuite/ directories

Configuration Checklist

Make sure that these tasks have been completed:

1. `config/configname` exists and is a directory
2. `config/configname/bli_config.h` exists and contains the proper definitions
3. `config/configname/bli_kernel.h` exists and contains the proper kernel definitions
4. `config/configname/make_defs.mk` exists and contains the desired build definitions
5. `config/configname/kernel_dir` exists and is a symbolic link (or actual directory) of kernels *and* kernel headers (not necessary for reference implementation)

make configuration

Simply run:

```
$ ./configure <configname>
```

where `<configname>` is the configuration sub-directory name chosen in Step 1 (defaults to `reference`)

Compilation

Simply run:

```
$ make
```

To see individual command line invocations, edit `make_defs.mk` with

```
BLIS_ENABLE_VERBOSE_MAKE_OUTPUT=yes
```

Installation

Simply run:

```
$ make install
```

The results in your PREFIX directory will look like:

```
# Check the contents of '<PREFIX>'.
$ ls -l /home/field/blis
lrwxrwxrwx 1 field dept    29 Dec  6 14:19 include -> include-0.0.1-4-reference
drwxr-xr-x 2 field dept 32768 Dec  6 14:19 include-0.0.1-4-reference
drwxr-xr-x 2 field dept  4096 Dec  6 14:19 lib

# Check the contents of '<PREFIX>/lib'.
$ ls -l /home/field/blis/lib
-rw-r--r-- 1 field dept 3919726 Dec  6 14:19 libblis-0.0.1-4-reference.a
lrwxrwxrwx 1 field dept      31 Dec  6 14:19 libblis.a -> libblis-0.0.1-4-reference.a
```

Outline

1. Requisites. Remote access
2. Building BLIS
   - Step 0: Obtaining BLIS
   - Step 1: Framework configuration
   - Step 2: `make` configuration
   - Step 3: Compilation
   - Step 4: Installation
3. Linking to BLIS
4. Building the BLIS Test Suite
5. Optimizing BLIS

Installation

Example:

```
BLIS_PREFIX = $(HOME)/blis
BLIS_INC    = $(BLIS_PREFIX)/include
BLIS_LIB    = $(BLIS_PREFIX)/lib/libblis.a

OTHER_LIBS  = -L/usr/lib -lm

CC          = gcc
CFLAGS      = -O2 -g -I$(BLIS_INC)
LINKER      = $(CC)

OBJS        = main.o util.o other.o

%.o: %.c
    $(CC) $(CFLAGS) -c $< -o $@

all: $(OBJS)
    $(LINKER) $(OBJS) $(BLIS_LIB) $(OTHER_LIBS) -o my_program.x
```

Outline

1. Requisites. Remote access

2. Building BLIS
   - Step 0: Obtaining BLIS
   - Step 1: Framework configuration
   - Step 2: `make` configuration
   - Step 3: Compilation
   - Step 4: Installation

3. Linking to BLIS

4. Building the BLIS Test Suite

5. Optimizing BLIS

BLIS Test Suite

- Complete functionality test for BLIS
- Configurable: operations, problem sizes, data types, data layout, error checking, . . .
- Directory `testsuite/`
- Configuration files:

  input.general Determine general test configuration
  input.operations Determine which operations to test
- Results can be directly processed by Matlab / Octave if instructed in `input.general`

Outline

1. Requisites. Remote access

2. Building BLIS
   - Step 0: Obtaining BLIS
   - Step 1: Framework configuration
   - Step 2: `make` configuration
   - Step 3: Compilation
   - Step 4: Installation

3. Linking to BLIS

4. Building the BLIS Test Suite

5. Optimizing BLIS

What's next?

Try to create a new BLIS configuration taking the reference one as a template
Create your own microkernels and enjoy!