Faq

Elethom edited this page 24 days ago · 70 revisions

General questions

OS and Compiler

General questions

What is BLAS? Why is it important?

BLAS stands for Basic Linear Algebra Subprograms. BLAS provides standard interfaces for linear algebra, including BLAS1 (vector-vector operations), BLAS2 (matrix-vector operations), and BLAS3 (matrix-matrix operations). In general, BLAS is the computational kernel ("the bottom of the food chain") in linear algebra or scientific applications. Thus, if BLAS implementation is highly optimized, the whole application can get substantial benefit.

What functions are there and how can I call them from my C code?

As BLAS is a standardized interface, you can refer to the documentation of its reference implementation at netlib.org. Calls from C go through its CBLAS interface, so your code will need to include the provided cblas.h in addition to linking with -lopenblas. A single-precision matrix multiplication will look like

```
#include <cblas.h>
...
cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, M, N, K, 1.0, A, K, B, N, 0.0,
result, N);
```

where M,N,K are the dimensions of your data - see https://petewarden.files.wordpress.com/2015/04/gemm_corrected.png (This image is part of an article on GEMM in the context of deep learning that is well worth reading in full - https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/)

What is OpenBLAS? Why did you create this project?

OpenBLAS is an open source BLAS library forked from the GotoBLAS2-1.13 BSD version. Since Mr. Kazushige Goto left TACC, GotoBLAS is no longer being maintained. Thus, we created this project to continue developing OpenBLAS/GotoBLAS.

What's the difference between OpenBLAS and GotoBLAS?

In OpenBLAS 0.2.0, we optimized level 3 BLAS on the Intel Sandy Bridge 64-bit OS. We obtained a performance comparable with that Intel MKL.

We optimized level 3 BLAS performance on the ICT Loongson-3A CPU. It outperformed GotoBLAS by 135% in a single thread and 120% in 4 threads.

We fixed some GotoBLAS bugs including a SEGFAULT bug on the new Linux kernel, MingW32/64 bugs, and a ztrmm computing error bug on Intel Nehalem.

We also added some minor features, e.g. supporting "make install", compiling without LAPACK and upgrading the LAPACK version to 3.4.2.

You can find the full list of modifications in Changelog.txt.

Where do parameters GEMM_P, GEMM_Q, GEMM_R come from?

The detailed explanation is probably in the original publication authored by Kazushige Goto - Goto, Kazushige; van de Geijn, Robert A; Anatomy of high-performance matrix multiplication. ACM Transactions on Mathematical Software (TOMS). Volume 34 Issue 3, May 2008 While this article is paywalled and too old for preprints to be available on arxiv.org, more recent publications like https://arxiv.org/pdf/1609.00076 contain at least a brief description of the algorithm. In practice, the values are derived by experimentation to yield the block sizes that give the highest performance. A general rule of thumb for selecting a starting point seems to be that PxQ is about half the size of L2 cache.

How can I report a bug?

Please file an issue at this issue page or send mail to the OpenBLAS mailing list.

Please provide the following information: CPU, OS, compiler, and OpenBLAS compiling flags (Makefile.rule). In addition, please describe how to reproduce this bug.

How to reference OpenBLAS.

You can reference our papers in this page. Alternatively, you can cite the OpenBLAS homepage http://www.openblas.net.

How can I use OpenBLAS in multi-threaded applications?

If your application is already multi-threaded, it will conflict with OpenBLAS multi-threading. Thus, you must set OpenBLAS to use single thread as following.

- export OPENBLAS_NUM_THREADS=1 in the environment variables. Or
- Call openblas_set_num_threads(1) in the application on runtime. Or
- Build OpenBLAS single thread version, e.g. make USE_THREAD=0 USE_LOCKING=1 (see comment below)

If the application is parallelized by OpenMP, please build OpenBLAS with USE_OPENMP=1

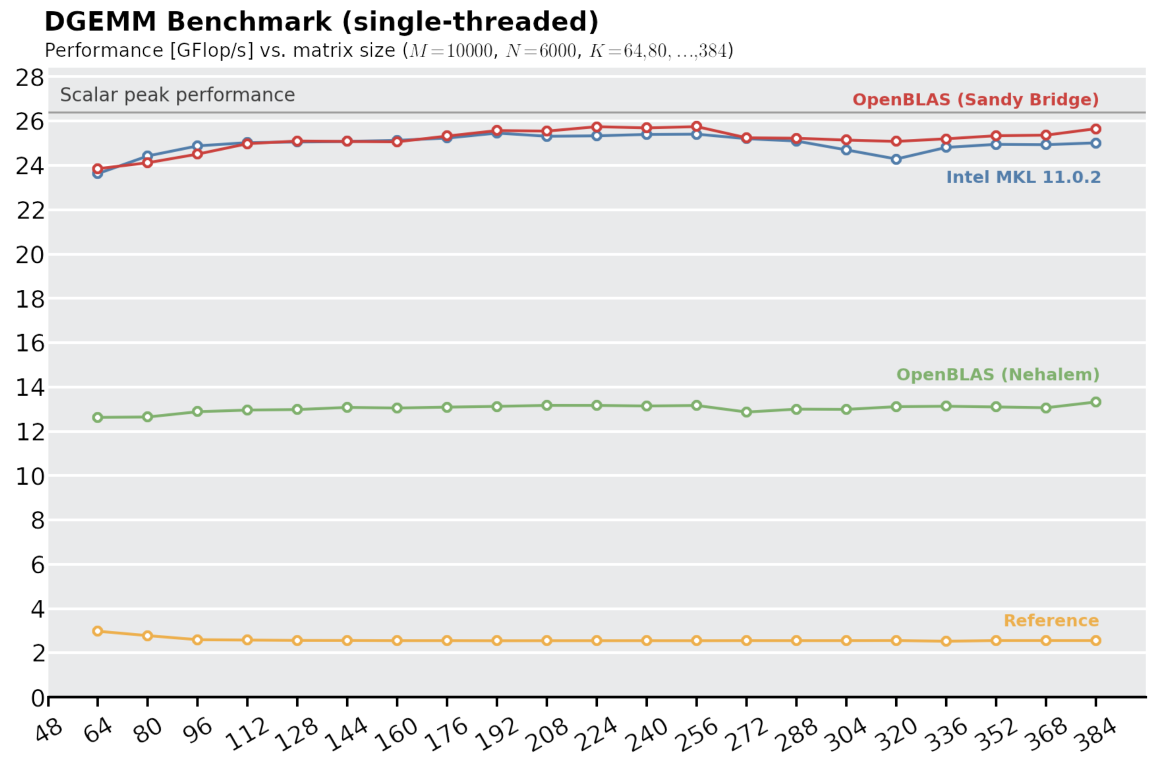
With the increased availability of fast multicore hardware it has unfortunately become clear that the thread management provided by OpenMP is not sufficient to prevent race conditions when OpenBLAS was built single-threaded by USE_THREAD=0 and there are concurrent calls from multiple threads to OpenBLAS functions. In this case, it is vital to also specify USE_LOCKING=1 (introduced with OpenBLAS 0.3.7).

What support is there for recent PC hardware ? What about GPU ?

- As OpenBLAS is a volunteer project, it can take some time for the combination of a capable developer, free time, and particular hardware to come along, even for relatively common processors. Starting from 0.3.1, support is being added for AVX 512 (TARGET=SKYLAKEX), requiring a compiler that is capable of handling avx512 intrinsics. While AMD Zen processors should be autodetected by the build system, as of 0.3.2 they are still handled exactly like Intel Haswell. There once was an effort to build an OpenCL implementation that one can still find at https://github.com/xianyi/clOpenBLAS , but work on this stopped in 2015.

How about the level 3 BLAS performance on Intel Sandy Bridge?

We obtained a performance comparable with Intel MKL that actually outperformed Intel MKL in some cases. Here is the result of the DGEMM subroutine's performance on Intel Core i5-2500K Windows 7 SP1 64-bit:



OS and Compiler

How can I call an OpenBLAS function in Microsoft Visual Studio?

Please read this page.

How can I use CBLAS and LAPACKE without C99 complex number support (e.g. in Visual Studio)?

Zaheer has fixed this bug. You can now use the structure instead of C99 complex numbers. Please read this issue page for details.

This issue is for using LAPACKE in Visual Studio.

I get a SEGFAULT with multi-threading on Linux. What's wrong?

This may be related to a bug in the Linux kernel 2.6.32 (?). Try applying the patch segaults.patch to disable mbind using

```
patch < segfaults.patch
```

and see if the crashes persist. Note that this patch will lead to many compiler warnings.

When I make the library, there is no such instruction: `xgetbv' error. What's wrong?

Please use GCC 4.4 and later version. This version supports xgetbv instruction. If you use the library for Sandy Bridge with AVX instructions, you should use GCC 4.6 and later version.

On Mac OS X, please use Clang 3.1 and later version. For example, make CC=clang

For the compatibility with old compilers (GCC < 4.4), you can enable NO_AVX flag. For example, make NO_AVX=1

My build fails due to the linker error "multiple definition of `dlamc3_'". What is the problem?

This linker error occurs if GNU patch is missing or if our patch for LAPACK fails to apply.

Background: OpenBLAS implements optimized versions of some LAPACK functions, so we need to disable the reference versions. If this process fails we end with duplicated implementations of the same function.

My build worked fine and passed all tests, but running `make lapack-test` ends with segfaults

Some of the LAPACK tests, notably in xeigtstz, try to allocate around 10MB on the stack. You may need to use `ulimit -s` to change the default limits on your system to allow this.

How could I disable OpenBLAS threading affinity on runtime?

You can define the OPENBLAS_MAIN_FREE or GOTOBLAS_MAIN_FREE environment variable to disable threading affinity on runtime. For example, before the running,

```
export OPENBLAS_MAIN_FREE=1
```

Alternatively, you can disable affinity feature with enabling NO_AFFINITY=1 in Makefile.rule.

How to solve undefined reference errors when statically linking against libopenblas.a

On Linux, if OpenBLAS was compiled with threading support (`USE_THREAD=1` by default), custom programs statically linked against `libopenblas.a` should also link to the pthread library e.g.:

```
gcc -static -I/opt/OpenBLAS/include -L/opt/OpenBLAS/lib -o my_program my_program.c -
lopenblas -lpthread
```

Failing to add the `-lpthread` flag will cause errors such as:

```
/opt/OpenBLAS/libopenblas.a(memory.o): In function `_touch_memory':
memory.c:(.text+0x15): undefined reference to `pthread_mutex_lock'
memory.c:(.text+0x41): undefined reference to `pthread_mutex_unlock'
/opt/OpenBLAS/libopenblas.a(memory.o): In function `openblas_fork_handler':
memory.c:(.text+0x440): undefined reference to `pthread_atfork'
/opt/OpenBLAS/libopenblas.a(memory.o): In function `blas_memory_alloc':
memory.c:(.text+0x7a5): undefined reference to `pthread_mutex_lock'
memory.c:(.text+0x825): undefined reference to `pthread_mutex_unlock'
/opt/OpenBLAS/libopenblas.a(memory.o): In function `blas_shutdown':
memory.c:(.text+0x9e1): undefined reference to `pthread_mutex_lock'
memory.c:(.text+0xa6e): undefined reference to `pthread_mutex_unlock'
/opt/OpenBLAS/libopenblas.a(blas_server.o): In function `blas_thread_server':
blas_server.c:(.text+0x273): undefined reference to `pthread_mutex_lock'
blas_server.c:(.text+0x287): undefined reference to `pthread_mutex_unlock'
blas_server.c:(.text+0x33f): undefined reference to `pthread_cond_wait'
/opt/OpenBLAS/libopenblas.a(blas_server.o): In function `blas_thread_init':
blas_server.c:(.text+0x416): undefined reference to `pthread_mutex_lock'
blas_server.c:(.text+0x4be): undefined reference to `pthread_mutex_init'
blas_server.c:(.text+0x4ca): undefined reference to `pthread_cond_init'
blas_server.c:(.text+0x4e0): undefined reference to `pthread_create'
blas_server.c:(.text+0x50f): undefined reference to `pthread_mutex_unlock'
...
```

The `-lpthread` is not required when linking dynamically against `libopenblas.so.0` .

Building OpenBLAS for Haswell or Dynamic Arch on RHEL-6, CentOS-6, Rocks-6.1,Scientific Linux 6

Minimum requirement to actually run AVX2-enabled software like OpenBLAS is kernel-2.6.32-358, shipped with EL6U4 in 2013

The `binutils` package from RHEL6 does not know the instruction `vpermpd` or any other AVX2 instruction. You can download a newer `binutils` package from Enterprise Linux software collections, following instructions here:
https://www.softwarecollections.org/en/scls/rhscl/devtoolset-3/
After configuring repository you need to install devtoolset-?-binutils to get later usable binutils package

```
$ yum search devtoolset-\?-binutils
$ sudo yum install devtoolset-3-binutils
```

once packages are installed check the correct name for SCL redirection set to enable new version

```
$ scl --list
devtoolset-3
rh-python35
```

Now just prefix your build commands with respective redirection:

```
$ scl enable devtoolset-3 -- make DYNAMIC_ARCH=1
```

AVX-512 (SKYLAKEX) support requires binutils, gcc and gcc-fortran from devtoolset-4 or later and kernel 2.6.32-696 aka 6U9 or 3.10.0-327 aka 7U2 or later to run. In absence of abovementioned toolset OpenBLAS will fall back to AVX2 instructions in place of AVX512 sacrificing some performance on SKYLAKE-X platform

Building OpenBLAS in QEMU/KVM

By default, QEMU reports the CPU as "QEMU Virtual CPU version 2.2.0", which shares CPUID with existing 32bit CPU even in 64bit virtual machine, and OpenBLAS recognizes it as PENTIUM2. Depending on the exact combination of CPU features the hypervisor choses to expose, this may not correspond to any CPU that exists, and OpenBLAS will error when trying to build. To fix this, pass `-cpu host` or `-cpu passthough` to QEMU, or another CPU model.

Building OpenBLAS for MIPS

For mips targets you will need latest toolchains P5600 - MTI GNU/Linux Toolchain I6400, P6600 - IMG GNU/Linux Toolchain

The download link is below (http://codescape-mips-sdk.imgtec.com/components/toolchain/2016.05-03/downloads.html)

You can use following commandlines for builds

```
IMG_TOOLCHAIN_DIR={full IMG GNU/Linux Toolchain path including "bin" directory -- for
example, /opt/linux_toolchain/bin}
IMG_GCC_PREFIX=mips-img-linux-gnu
IMG_TOOLCHAIN=${IMG_TOOLCHAIN_DIR}/${IMG_GCC_PREFIX}

I6400 Build (n32):
make BINARY=32 BINARY32=1 CC=$IMG_TOOLCHAIN-gcc AR=$IMG_TOOLCHAIN-ar FC="$IMG_TOOLCHAIN-
gfortran -EL -mabi=n32" RANLIB=$IMG_TOOLCHAIN-ranlib HOSTCC=gcc CFLAGS="-EL"
FFLAGS=$CFLAGS LDFLAGS=$CFLAGS TARGET=I6400

I6400 Build (n64):
make BINARY=64 BINARY64=1 CC=$IMG_TOOLCHAIN-gcc AR=$IMG_TOOLCHAIN-ar FC="$IMG_TOOLCHAIN-
gfortran -EL" RANLIB=$IMG_TOOLCHAIN-ranlib HOSTCC=gcc CFLAGS="-EL" FFLAGS=$CFLAGS
LDFLAGS=$CFLAGS TARGET=I6400

P6600 Build (n32):
make BINARY=32 BINARY32=1 CC=$IMG_TOOLCHAIN-gcc AR=$IMG_TOOLCHAIN-ar FC="$IMG_TOOLCHAIN-
gfortran -EL -mabi=n32" RANLIB=$IMG_TOOLCHAIN-ranlib HOSTCC=gcc CFLAGS="-EL"
FFLAGS=$CFLAGS LDFLAGS=$CFLAGS TARGET=P6600

P6600 Build (n64):
make BINARY=64 BINARY64=1 CC=$IMG_TOOLCHAIN-gcc AR=$IMG_TOOLCHAIN-ar FC="$IMG_TOOLCHAIN-
gfortran -EL" RANLIB=$IMG_TOOLCHAIN-ranlib HOSTCC=gcc CFLAGS="-EL" FFLAGS="$CFLAGS"
LDFLAGS="$CFLAGS" TARGET=P6600

MTI_TOOLCHAIN_DIR={full MTI GNU/Linux Toolchain path including "bin" directory -- for
example, /opt/linux_toolchain/bin}
MTI_GCC_PREFIX=mips-mti-linux-gnu
MTI_TOOLCHAIN=${IMG_TOOLCHAIN_DIR}/${IMG_GCC_PREFIX}

P5600 Build:

make BINARY=32 BINARY32=1 CC=$MTI_TOOLCHAIN-gcc AR=$MTI_TOOLCHAIN-ar FC="$MTI_TOOLCHAIN-
gfortran -EL"    RANLIB=$MTI_TOOLCHAIN-ranlib HOSTCC=gcc CFLAGS="-EL" FFLAGS=$CFLAGS
LDFLAGS=$CFLAGS TARGET=P5600
```

Building OpenBLAS on POWER fails with IBM XL

```
Trying to compile OpenBLAS with IBM XL ends with error messages about unknown register
names
```

like "vs32". Working around these by using known alternate names for the vector registers only leads to another assembler error about unsupported constraints. This is a known deficiency in the IBM compiler at least up to and including 16.1.0 (and in the POWER version of clang, from which it is derived) - use gcc instead. (See issues #1078 and #1699 for related discussions)

Replacing system BLAS/updating APT OpenBLAS in Mint/Ubuntu/Debian

Debian and Ubuntu LTS versions provide OpenBLAS package which is not updated after initial release, and under circumstances one might want to use more recent version of OpenBLAS e.g. to get support for newer CPUs

Ubuntu and Debian provides 'alternatives' mechanism to comfortably replace BLAS and LAPACK libraries systemwide.

After successful build of OpenBLAS (with DYNAMIC_ARCH set to 1)

```
$ make clean
$ make DYNAMIC_ARCH=1
$ sudo make DYNAMIC_ARCH=1 install
```

One can redirect BLAS and LAPACK alternatives to point to source-built OpenBLAS First you have to install NetLib LAPACK reference implementation (to have alternatives to replace):

```
$ sudo apt install libblas-dev liblapack-dev
```

Then we can set alternative to our freshly-built library:

```
$ sudo update-alternatives --install /usr/lib/libblas.so.3 libblas.so.3
/opt/OpenBLAS/lib/libopenblas.so.0 41 \
   --slave /usr/lib/liblapack.so.3 liblapack.so.3 /opt/OpenBLAS/lib/libopenblas.so.0
```

Or remove redirection and switch back to APT-provided BLAS implementation order:

```
$ sudo update-alternatives --remove libblas.so.3 /opt/OpenBLAS/lib/libopenblas.so.0
```

In recent versions of the distributions, the installation path for the libraries has been changed to include the name of the host architecture, like /usr/lib/x86_64-linux-gnu/blas/libblas.so.3 or libblas.so.3.x86_64-linux-gnu. Use `$ update-alternatives --display libblas.so.3` to find out what layout your system has.

I built OpenBLAS for use with some other software, but that software cannot find it

Openblas installs as a single library named libopenblas.so, while some programs may be searching for a separate libblas.so and liblapack.so so you may need to create appropriate symbolic links (`ln -s libopenblas.so libblas.so; ln -s libopenblas.so liblapack.so`) or copies. Also make sure that the installation location (usually /opt/OpenBLAS/lib or /usr/local/lib) is among the library search paths of your system.

I included cblas.h in my program, but the compiler complains about a missing common.h or functions from it

You probably tried to include a cblas.h that you simply copied from the OpenBLAS source, instead you need to run `make install` after building OpenBLAS and then use the modified cblas.h that this step builds in the installation path (usually either /usr/local/include, /opt/OpenBLAS/include or whatever you specified as PREFIX= on the `make install`)

Compiling OpenBLAS with gcc's -fbounds-check actually triggers aborts in programs

This is due to different interpretations of the (informal) standard for passing characters as arguments between C and FORTRAN functions. As the method for storing text differs in the two languages, when C calls Fortran the text length is passed as an "invisible" additional parameter. Historically, this has not been required when the text is just a single character, so older code like the Reference-LAPACK bundled with OpenBLAS does not do it. Recently gcc's checking has changed to require it, but there is no consensus yet if and how the existing LAPACK (and many other codebases) should adapt. (And for actual compilation, gcc has mostly backtracked and provided compatibility options - hence the default build settings in the OpenBLAS Makefiles add -fno-optimize-sibling-calls to the gfortran options to prevent miscompilation with "affected" versions. See ticket 2154 in the issue tracker for more details and links)

Build fails with lots of errors about undefined ?GEMM_UNROLL_M

Your cpu is apparently too new to be recognized by the build scripts, so they failed to assign appropriate parameters for the block algorithm. Do a `make clean` and try again with TARGET set to one of the cpu models listed in `TargetList.txt` - for x86_64 this will usually be HASWELL.

CMAKE/OSX: Build fails with 'argument list too long'

This is a limitation in the maximum length of a command on OSX, coupled with how CMAKE works. You should be able to work around this by adding the option `-DCMAKE_Fortran_USE_RESPONSE_FILE_FOR_OBJECTS=1` to your CMAKE arguments.

Usage

Program is Terminated. Because you tried to allocate too many memory regions

In OpenBLAS, we mange a pool of memory buffers and allocate the number of buffers as the following.

```
#define NUM_BUFFERS (MAX_CPU_NUMBER * 2)
```

This error indicates that the program exceeded the number of buffers.

Please build OpenBLAS with larger `NUM_THREADS`. For example, `make NUM_THREADS=32` or `make NUM_THREADS=64`. In `Makefile.system`, we will set `MAX_CPU_NUMBER=NUM_THREADS`.

How to choose TARGET manually at runtime when compiled with DYNAMIC_ARCH

The environment variable which control the kernel selection is `OPENBLAS_CORETYPE` (see `driver/others/dynamic.c`) e.g. `export OPENBLAS_CORETYPE=Haswell`. And the function `char* openblas_get_corename()` returns the used target.

After updating the installed OpenBLAS, a program complains about "undefined symbol gotoblas"

This symbol gets defined only when OpenBLAS is built with "make DYNAMIC_ARCH=1" (which is what distributors will choose to ensure support for more than just one CPU type).

How can I find out at runtime what options the library was built with ?

OpenBLAS has two utility functions that may come in here:

openblas_get_parallel() will return 0 for a single-threaded library, 1 if multithreading without OpenMP, 2 if built with USE_OPENMP=1

openblas_get_config() will return a string containing settings such as USE64BITINT or DYNAMIC_ARCH that were active at build time, as well as the target cpu (or in case of a dynamic_arch build, the currently detected one).

After making OpenBLAS, I find that the static library is multithreaded, but the dynamic one is not ?

The shared OpenBLAS library you built is probably working fine as well, but your program may be picking up a different (probably single-threaded) version from one of the standard system paths like /usr/lib on startup. Running `ldd /path/to/your/program` will tell you which library the linkage loader will actually use.

Specifying the "correct" library location with the `-L` flag (like `-L /opt/OpenBLAS/lib`) when linking your program only defines which library will be used to see if all symbols *can* be resolved, you will need to add an rpath entry to the binary (using `-Wl,rpath=/opt/OpenBLAS/lib`) to make it request searching that location. Alternatively, remove the "wrong old" library (if you can), or set LD_LIBRARY_PATH to the desired location before running your program.

I want to use OpenBLAS with CUDA in the HPL 2.3 benchmark code but it keeps looking for Intel MKL

You need to edit file src/cuda/cuda_dgemm.c in the NVIDIA version of HPL, change the "handle2" and "handle" dlopen calls to use libopenblas.so instead of libmkl_intel_lp64.so, and add an trailing underscore in the dlsym lines for dgemm_mkl and dtrsm_mkl (like `dgemm_mkl = (void(*)())dlsym(handle, “dgemm_”);`)

Multithreaded OpenBLAS runs no faster or is even slower than singlethreaded on my ARMV7 board

The power saving mechanisms of your board may have shut down some cores, making them invisible to OpenBLAS in its startup phase. Try bringing them online before starting your calculation.

Speed varies wildly between individual runs on a typical ARMV8 smartphone processor

Check the technical specifications, it could be that the SoC combines fast and slow cpus and threads can end up on either. In that case, binding the process to specific cores e.g. by setting `OMP_PLACES=cores` may help. (You may need to experiment with OpenMP options, it has been reported that using `OMP_NUM_THREADS=2 OMP_PLACES=cores` caused a huge drop in performance on a 4+4 core chip while `OMP_NUM_THREADS=2 OMP_PLACES=cores(2)` worked as intended - as did OMP_PLACES=cores with 4 threads)

I cannot get OpenBLAS to use more than a small subset of available cores on a big system

Multithreading support in OpenBLAS requires the use of internal buffers for sharing partial results, the number and size of which is defined at compile time. Unless you specify NUM_THREADS in your make or cmake command, the build scripts try to autodetect the number of cores available in your build host to size the library to match. This unfortunately means that if you move the resulting binary from a small "front-end node" to a larger "compute node" later, it will still be limited to the hardware capabilities of the original system. The solution is to set NUM_THREADS to a number big enough to encompass the biggest systems you expect to run the binary on - at runtime, it will scale down the maximum number of threads it uses to match the number of cores physically available.

+ Add a custom footer

Pages 21

Find a Page…

+ Add a custom sidebar

Clone this wiki locally

```
https://github.com/xianyi/OpenBLAS.wiki.git
```